68' MICRO JOURNAL

Australia A $4.75 | New Zealand NZ $ 6.50
Singapore S $9.45 | Hong Kong H $23.50
Malaysia M $9.45 | Sweden 30 SEK

$2.95 USA

Motorola VME-MACINTOSH-S 50 & Other 68XXX Systems

6809 68008 68000 68010 68020 68030

OS-9

The Magazine for Motorola CPU Devices
For Over a Decade!
A User Contributor Journal

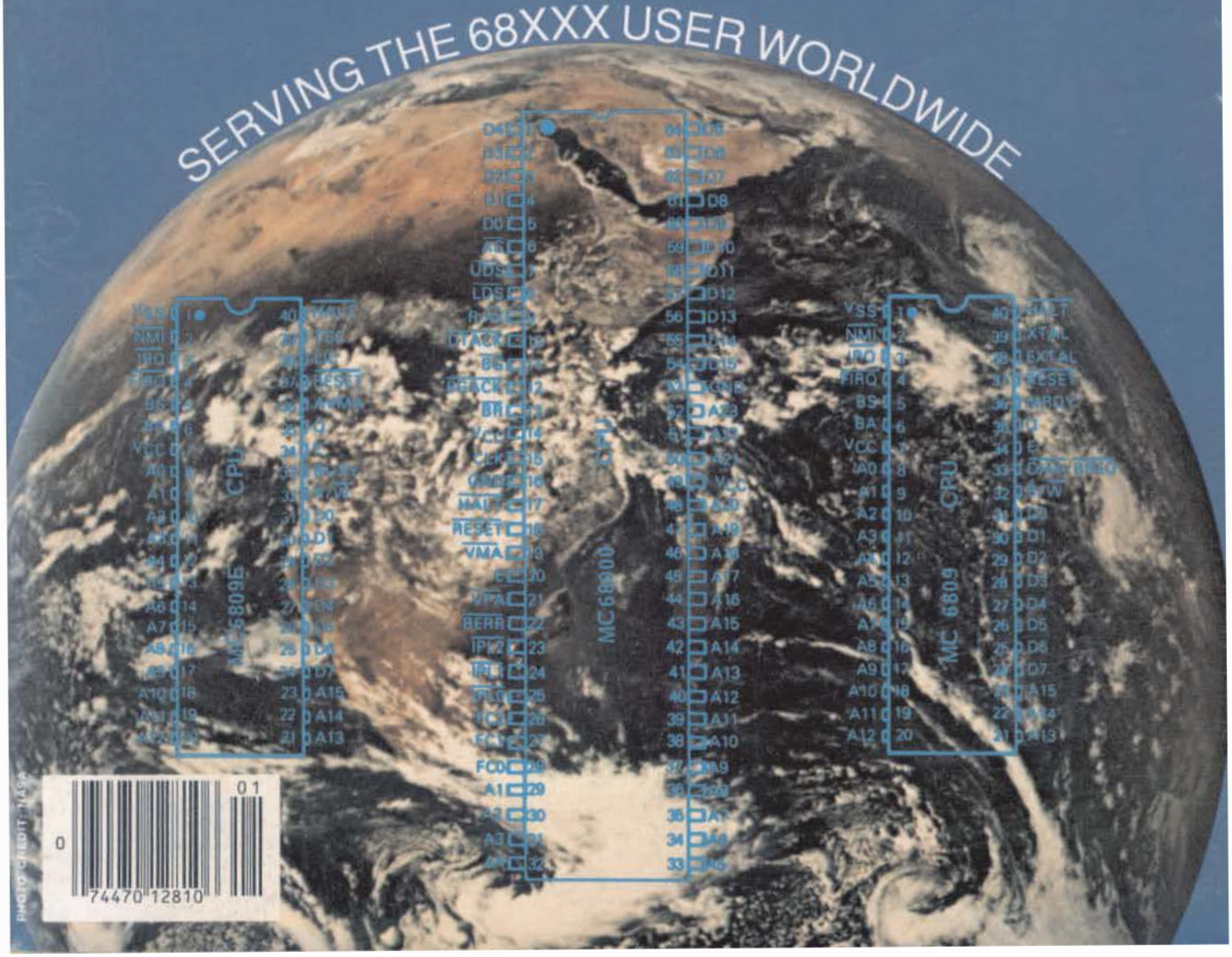
FLEX SK*DOS

This Issue:

And Lots More!

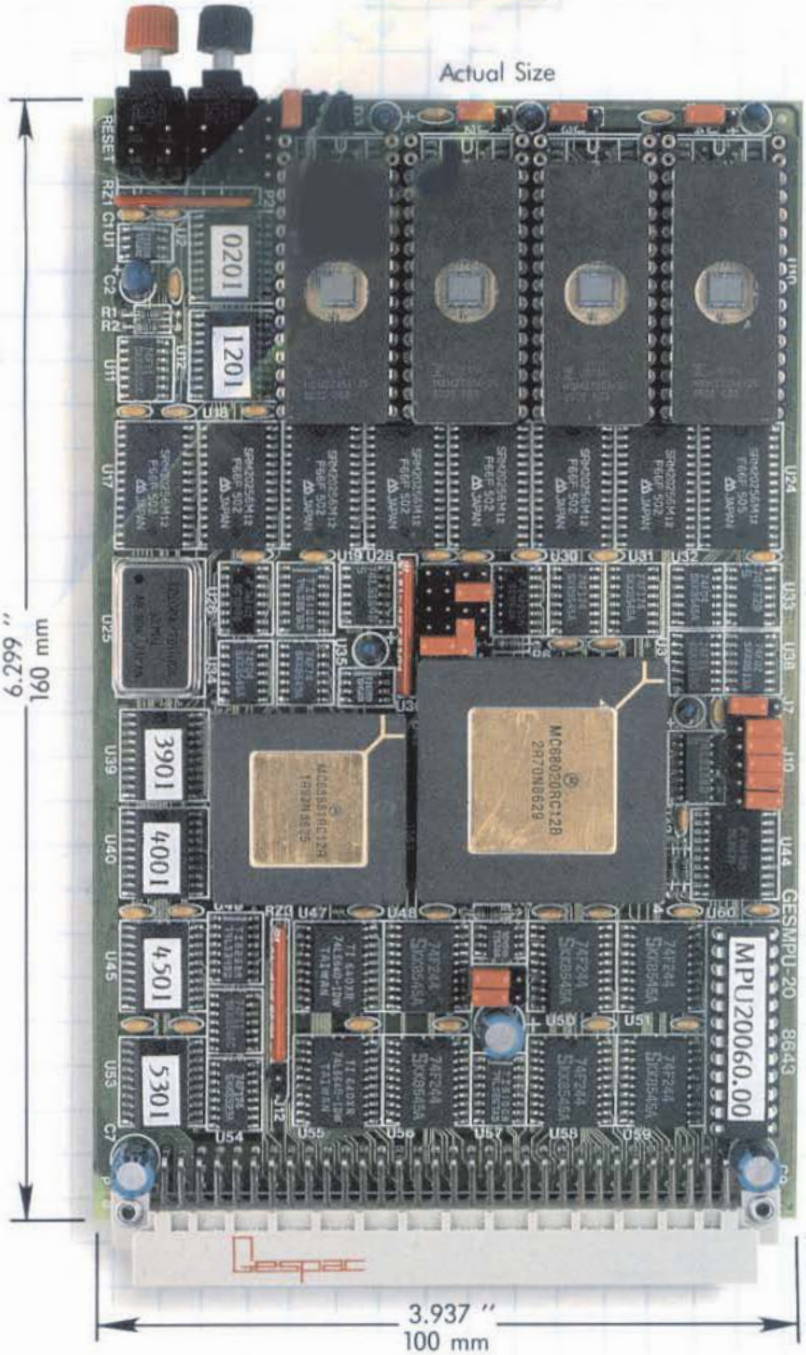
VOLUME IX ISSUE 1 ● Devoted to the 68XXX User ● January 1987

"Small Computers Doing Big Things"



0 74470 12810 01

GESPAC Gives You More Power Per Square Inch.



Here is the size, performance, and cost breakthrough you have been waiting for: The 68020 based GESMPU-20 from GESPAC.

You can now unleash an unprecedented amount of power into your applications. On just 25 square inches, we have squeezed a 12.5 MHz (16 MHz optional) 68020 32-bit microprocessor, a 68881 floating point coprocessor, 4 sockets for up to 512 Kilobytes of EPROM, and up to 512 Kilobytes of zero-wait-states CMOS RAM.

This board is totally expandable through its G-64 bus interface. And GESPAC has the largest variety of inexpensive memory, interfaces, controllers, and transducer cords anywhere. Plus real-time disk operating systems, high level languages, and other software tools. GESPAC has the total solution to your system integration needs.

Best of all, because our boards are small, they cost less. The new GESMPU-20 is priced below $1000 in one hundred piece quantity orders.

So why wait? Contact us today for information on the GESMPU-20 or any of the 150 G-64 bus system components from GESPAC—the leader in single Eurocard microcomputer products worldwide.

Call (602) 962-5559.

Gespac

IN USA - CANADA
50A West Hoover Ave.
Mesa, Arizona 85202
Tel. (602) 962-5559
Telex 386575

INTERNATIONAL
3, chemin des Aulx
CH-1228 Geneva
Tel. (022) 713400
Telex 429989

BOARDS & PARTS FOR GIMIX SYSTEMS

CONTACT GIMIX FOR PRICES, DETAILS, AND REQUIRMENTS FOR HARD DISK AND OTHER MASS STORAGE UPGRADES.

THE GIMIX CLASSY CHASSIS #19 consists of a heavyweight aluminum cabinet, constant voltage ferro-resonant power supply, and SS50 Mother board with baud rate generator board $1498.19
#22 Triple Disk Regulator Card $88.22
#93 Baud Rate Generator Board $88.93
#23 Missing Cycle Detector $36.23
#92 Filler Plate ..... $14.92 50 Hz Option $30.00
Cable sets 8" with Back Panel connector $29.25
for two 8" external drives ..... $44.26 for two 5" drives $34.98

CPU BOARDS

#01 GMX III CPU & OS-9 GMX III $1698.01
#02 GMX IIICPU & UniFLEX III $1998.02
The #05 GIMIX 6809 PLUS CPU Board $578.05
Options: GIMIX DAT ..... $35.00 9511A $312.00
SWTP DAT ..... $15.00 9512 ..... $265.00
#03 6800 CPU $224.03
#06 6800 CPU w/Timers $288.08
6800 Baud Rate Option ..... Add $30.00

FLOPPY DISK CONTROLLER

#68 DMA $588.68

MEMORY BOARDS FOR 6809/68020 SYSTEMS

#72 256KB CMOS SIATIC RAM board with battery back up (specify system) $648.72

MEMORY BOARDS (6800/6809 SYSTEMS ONLY)

#34 8K PROM Card $98.34
#32 16 Socket PROM/ROM/RAM Board, 24 pin $238.32
#31 16 Socket Universal Memory Board, 24/28 pin $268.31

INTELLIGENT I/O PROCESSOR BOARDS

significantly reduce systems overhead by handling routine I/O functions, freeing the host CPU for running user programs. This improves overall system performance and allows user terminals to be run at up to 19.2K baud. For use with GMX III and 020 systems.

#11 3 Port Serial-30 Pin (OS9) $498.11
#14 3 Port Serial-30 Pin (UniFLEX) $498.14
#12 Parallel-50 Pin (UniFLEX-020) $538.12
#134 Port Serial-50 Pin (OS9 & UniFLEX-020) $618.13
#15 24K Version of #11, with either large input or output buffers (specify) $648.15

I/O BOARDS (6800/6809 SYSTEMS ONLY)

#41 Serial, 1 Port $88.41
#43 Serial, 2 Port $128.43
#46 Serial, 8 Port (OS9/FLEX only) $318.48
#42 Parallel, 2 Port $88.42
#44 Parallel, 2 Port (Centronics pinout) $128.44
#45 Parallel, 8 Port (OS9/FLEX only) $198.45
#501/Olor RS-232C, 423, 422-w/6850 ..... $244.50
#52 SSDA with 6852 ..... $254.52
#54 ADLC with 6854 ..... $268.54

CABLES FOR I/O BOARDS — SPECIFY BOARD

#95 Cable sets (1 needed per port) $24.95
#51 Cent. 8 P. Cable for #12 & #44 ..... $34.51
#53 Cent. Cable Set $36.53

OTHER BOARDS & PARTS

#66 Prototyping Board-50 Pin $56.66
#33 Prototyping Board-30 Pin $38.33
Windiush EPROM Programmer S30 (OS9/FLEX 6809 only) $545.00
#76 Video Board-80 x 24 ..... $398.76
#08 Relay Driver Package $1128.08
#86 Above without Relays $538.86
Opto Board $348.85
Binder, 3" $12.00
Binder, 2" $9.00

8" DRIVE CABINET & PARTS

2 8" DSDD Drives, Cabinet & Cables 60 Hz only $1598.68
Cabinet Only for 8" Drive $648.18
220v/50 Hz Option Add $30.00
Cable Set-Internal for 2 Drives $44.82
Cable Set-Internal for 4 Drives $67.84
Cable from 8" Cab. to Mainframe $45.81
8" Filler Plate $14.83

SOFTWARE:

GIMIX exclusive versions of OS-9/GMX I, II, III & FLEX are for GIMIX hardware only. All versions of OS-9 require the #68 controller. When ordered with controller, FLEX is $30.00
GIMIX versions of FLEX $90.00
GMX VDisk for FLEX 09 ..... $100.00
GMXBUG: PROMs & Manual $148.65
Boot or Video/Boot PROMS (6809) $30.00
GIMIX Boot PROM for UNIFLEX $50.00
RMS (OS9) $250.00
DO (OS9) $70.00
OS-9 GMX III Update w/CPU SPPTROM $125.00
I/O PROMS w/Update $40.00
GMXBUG/FLEX/VDISK w/OS-9 III update ..... Add $175.00
RAM Disk for OS-9 ..... $125.00
O-FLEX $250.00
OS9 GMX I $250.00
OS9 GMX II $500.00
SCULPTOR-6809 (UniFLEX/OS-9) $995.00
SCULPTOR—68020 (UniFLEX) $1595.00

CONTACT GIMIX FOR PRICES AND AVAILABILITY OF OPTIONAL UNIFLEX AND OS9 LANGUAGES AND OTHER SOFTWARE.

ALL PRICES ARE F.O.B. CHICAGO.

GIMIX DOES NOT GUARANTEE PERFORMANCE OF ANY GIMIX SYSTEMS, BOARDS OR SOFTWARE WHEN USED WITH OTHER MANUFACTURERS PRODUCT.

LIMITED WARRANTY

GMX INC. ("GMX") Warrants its products against defects in material and workmanship for a period of ninety days from the date of shipment. The obligation of GMX is limited to the repair or replacement of any product, free of all charges, which proves defective during this period. This warranty does not cover damage due to accidents, negligence, abuse or tampering.

GMX MAKES NO OTHER WARRANTIES OR GUARANTEES, EXPRESS, STATUTORY, OR IMPLIED, OF ANY KIND WHATSOEVER WITH RESPECT TO ANY PRODUCT PURCHASED, AND ANY IMPLIED WARRANTY OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE IS HEREBY DISCLAIMED BY GMX AND EXCLUDED FROM ANY AGREEMENT MADE BY GMX.

GMX will not be responsible for any damage of any kind not covered by the exclusive remedies set forth in this limited warranty. GMX will not be responsible for any special, indirect, or consequential damage caused by its products.

GMX products are not for consumer use. GMX expressly disclaims all warranties on any of its products which may be included in any product normally used for personal or family purposes.

Contact GMX by mail at 1337 West 37th Place, Chicago, IL 60609; or phone at (312) 927-5510; if your product is defective to arrange for its repair or replacement under this warranty.

Repair charges for GIMIX products after warranty period will be $35.00 per hour per board (minimum $35.00) plus parts. Customer pays freight charges both ways. If GIMIX determines that replacement is desirable instead, we will notify you. Charges for checking out complete system will be $500.00 plus parts, freight, and necessary board repairs.

GIMIX, Inc. reserves the right to change pricing, terms, and products specifications at any time without further notice.

GIMIX INC. 1337 WEST 37th PLACE • CHICAGO, ILLINOIS 60609
(312) 927-5510 • TWX 910-221-4055

A Member of the CPI Family

68 Micro Journal

6800 6809 68000 68010 68020

10 Years of Dedication to Motorola Users

Editorial Staff

Contents

"Contribute *Nothing* - Expect *Nothing*" DMW 1986

COMPUTER PUBLISHING, INC.
"Over a Decade of Service"
"World Wide"

68 MICRO JOURNAL
CPI
Computer Publishing Center
5900 Cassandra Smith Road
PO Box 849
Hixson, TN 37343

Phone (615) 842-4600 - Telex 510 600-6630

68 Micro Journal is published 12 times a year by Computer Publishing, Inc. Second Class Postage paid ISSN 0194-5025 at Hixson, TN and additional entries. Postmaster: send form 3597 to 68 Micro Journal, POB 849, Hixson, TN 37343.

Subscription Rates

1 Year: $24.50 USA, Canada & Mexico $34.00 *a year.* Others *add* $12.00 *a year surface, airmail add* $48.00 *a year, USA funds!* 2 Years $42.50, 3 Years $64.50 *plus additional postage, for each* additional year.

Items or Articles for Publication

Articles submitted for publication must include authors name, address, telephone number and date, as well as a statement that the material is original and the property of the submitting author. Articles submitted should be on diskette, Macintosh, OS-9 or FLEX format. All printed items should be dark type and satisfactory for photo-reproduction. No blue ink! No hand written articles - please.

Please do not *format with spaces any text indents, chart items, etc. (source listings o.k.) WE will edit in ALL formating.* Text should be flush left column and use ONLY a carriage return to separate paragraphs or other article text items! MacWrite, FLEX TSC, Stylo formatting acceptable.

Letters & Advertising Copy

Letters to the Editor should be original copy, signed! Letters of gripe as well as praise are acceptable. *We reserve the right to reject any letter to the editor or advertising copy material, for any reason we deem advisable.*

Advertising Rates: Commercial please contact 68 Micro Journal advertising department. Classified ads must be non-commercial. Minimum of $15.50 for first 15 words. Add $.60 per word after the first 15. All classifieds must be pre-paid. No classifieds accepted by telephone.

The VME BUS and OS-9:

Ultimate Software for the Ultimate Bus.

Modularity. Flexibility. High Performance. Future growth. These are probably the prime reasons you chose the VME bus. Why not use the same criteria when selecting your system software? That's why you should take a look at Microware's OS-9/68000 Operating System—it's the perfect match for the VME bus.

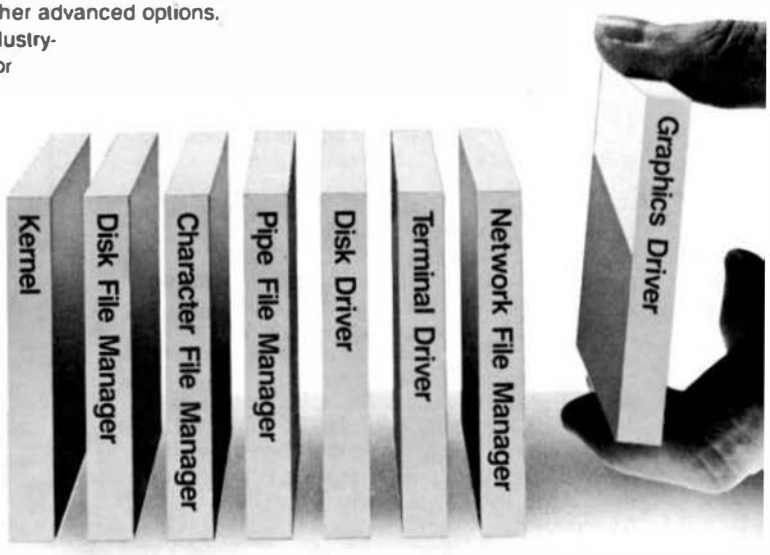
When you're working with VME you must have access to every part of the system. Unlike other operating systems that literally scream KEEP OUT!, OS-9's open architecture invites you to create, adapt, customize and expand. Thanks to its unique modular design, OS-9 naturally fits virtually any system, from simple ROM-based controllers up to large multiuser systems.

And that's just the beginning of the story. OS-9 gives you a complete UNIX-application compatible environment. It is multitasking, real time, and extremely fast. And if you're still not impressed, consider that a complete OS-9 executive and I/O driver package typically fits in less than 24K of RAM or ROM.

Software tools abound for OS-9, including outstanding Microware C, Basic, Fortran, and Pascal compilers. In addition, cross C compilers and cross assemblers are available for VAX systems under Unix or VMS. You can also plug in other advanced options, such as the GSS-DRIVERS™ Virtual Device Interface for industry-standard graphics support, or the OS-9 Network File Manager for high level, hardware-independent networking.

Designed for the most demanding OEM requirements, OS-9's performance and reliability has been proven in an incredible variety of applications. There's nothing like a track record as proof: to date, over 200 OEMs have shipped more than 100,000 OS-9-based systems.

Ask your VME system supplier about OS-9. Or you can install and evaluate OS-9 on your own custom system with a reasonably priced Microware PortPak™. Contact Microware today. We'll send you complete information about OS-9 and a list of quality manufacturers who offer off-the-shelf VME/OS-9 packages.



Modular Hardware Deserves Modular Software

MICROWARE.

Microware Systems Corporation
1866 N.W. 114th Street • Des Moines, Iowa 50322
Phone 515-224-1929 • Telex 910-520-2535

Microware Japan, Ltd.
41-19 Honcho 4-Chome, Funabashi City • Chiba 273, Japan • Phone 0473 (28) 4493 • Telex 781-299-3122

Micromaster Scandinavian AB
S:t Persgatan 7
Box 1309
S751-43 Uppsala
Sweden
Phone: 018-138595
Telex: 76129

Dr. Rudolf Keil, GmbH
Porphystrasse 15
D-6905 Schriesheim
West Germany
Phone: (0 62 03) 67 41
Telex: 465025

Elsoft AG
Zelweg 12
CH-5405 Baden-Dattwil
Switzerland
Phone: (056) 83-3377
Telex: 828275

Vivaway Ltd.
36-38 John Street
Luton, Bedfordshire, LU1 2JE
United Kingdom
Phone: (0582) 423425
Telex: 825115

Microprocessor Consultants
92 Bynya Road
Palm Beach 2108
NSW Australia
Phone: 02-919-4917

Microdata Soft
97 bis, rue de Colombes
92400 Courbevoie
France
Phone: 1-768-80-80
Telex: 615405

MUSTANG -020 Super SBC™

DATA-COMP proudly presents the first Under $5000 "SUPER MICRO".

The MUSTANG-020™

MUSTANG-020™

The MUSTANG-020 68020 SBC provides a powerful, compact, 32 bit computer system featuring the "state of the art" Motorola 68020 "super" micro-processor. It comes standard with 2 megabyte of high-speed SIP dynamic RAM, serial and parallel ports, floppy disk controller, a SASI hard disk interface for intelligent hard disk controllers and a battery backed-up time- f-day clock. Provisions are made for the super powerful Motorola MC68881 floating point math co-processor, for heavy math and number crunching applications. An optional network interface uses one serial (four (4) standard, expandable to 20) as a 125/bit per second network channel. Supports as many as 32 nodes.

The MUSTANG-020 is ideally suited to a wide variety of applications. It provides a cost effective alternative to the other MC68020 systems now available. It is an excellent introductory tool to the world of hi-power, hi-speed new generation "super micros". In practical applications it has numerous applications, ranging from scientific to education. It is already being used by government agencies, labs, universities, business and practically every other critical applications center, worldwide, where true multi-user, multi-tasking needs exist. The MUSTANG-020 is UNIX C level V compatible. Where low cost and power is a must, the MUSTANG-020 is the answer, as many have discovered. Proving that price is not the standard for quality!

As a software development station, a general purpose scientific or small to medium business computer, or a super efficient real-time con roller in process control, the MUSTANG-020 is the cost effective choice. With the optional MC68881 floating point math co-processor installed, it has the capability of systems costing many times over it's total acquisition cost.

With the DATA-COMP "total package", consisting of a heavy duty metal cabinet, switching power supply with rf/line by-passing, 5 inch DS/DD 80 track floppy, Xebec hard disk controller, 25 megabyte winchester hard disk, four serial RS-232 ports and a UNIX C level V compatible multi-tasking, multi-user operating system, the price is under $5000, w/12.5 megahertz system clock (limited time offer). Most all popular high level languages are available at very reasonable cost. The system is expandable to 20 serial ports, at a cost of less than $65 per port, in multiples of 8 port expansion options.

The system SBC fully populated, quality tested, with 4 serial ports pre-wired and board mounted is available for less that $3000. Quantity discounts are available for OEM and special applications, in quantity. All that is required to bring to complete "system" standards is a cabinet, power supply, disks and operating system. All these are available as separate items from DATA-COMP.

A special version of the Motorola 020-BUG is installed on each board. 020-BUG is a ROM based bebugger package with facilities for downloading and executing user programs from a host system. It includes commands for display and modification of memory, breakpoint capabilities, a powerful assembler/disassemble and numerous system diagnostics. Various 020-BUG system routines, such as I/O handlers are available for user programs.

Normal system speed is 3-4.5 MIPS, with burst up to 10 MIPS, at 16.6 megahertz. Intelligent I/O available for some operating systems.

Hands-on "actual experience sessions", before you buy, are available from DATA-COMP. Call or write for additional information or pricing.

DATA-COMP
Installed Systems World-Wide
OVER 10 YEARS OF DEDICATED QUALITY!
CPI A Division of Computer Publishing, Inc.
5900 Cassandra Smith Road
Hixson, Tn 37343
Telephone 615 842-4600
Telex 510 600-6630

MUSTANG-020, MUSTANG-08 Benchmarks

All timings by independent consultant

	Time - seconds 32 bit Integer	Register Long
IBM AT 7300 Xenix Sys 3	9.7	no register
AT&T 7300 UNIX PC 68010	7.2	4.3
DEC VAX 11/780 UNIX Berkley 4.2	3.6	3.2
DEC VAX 11/750 " "	5.1	3.2
68000 OS-9 68K 10 Mhz	6.5	4.0
68008 OS-9 68K 8 Mhz	18.0	9.0
MUSTANG-08 68008 OS-9 68K 10 Mhz	**9.8**	**6.3**
MUSTANG-020 68020 OS-9 68K 16 Mhz	**2.2**	**0.88**
MUSTANG-020 68020 MC68881 UniFLEX 16 Mhz	**1.8**	**1.22**

```
main()
    |
        register long i;
        for (i=0; i < 999999; ++i);
    |
```

Estimated MIPS - MUSTANG-020 --- 4.5 MIPS.
Burst to 8 - 10 MIPS: Motorola Specs

MUSTANG-020-

FEATURES

- 12.5 Mhz (optional) 16.6 Mhz available) MC68020 full 32-bit wide path processor
- 32-bit wide data and address buses, non-multiplexed
- on chip instruction cache
- object code compatible with all 68XXX family processors
- enhanced instruction set - math co-processor interface
- 68881 math hi-speed floating point co-processor (optional)
  direct extension of full 68020 instruction set
  full support IEEE P754, draft 10.0
  transcendental and other scientific math functions
- 2 Megabyte of SIP RAM (512 x 32 bit organization)
- up to 256K bytes of EPROM (64 x 32 bits)
- 4 Asychronous serial I/O ports standard
  optional to 20 serial ports
  standard RS-232 interface
  optional network interface
- buffered 8 bit parallel port (1/2 MC68230)
  Centronics type pinout
- expansion connector for additional I/O devices
  16 bit data path
  256 byte address space
  2 interrupt inputs
  clock and control signals
  Motorola I/O Channel Modules
- time of day clock/calendar w/battery backup
- controller for 2, 5 1/4" floppy disk drives
  single or double side, single or double density
  35 to 80 track selectable (48-96 TPI)
- SASI interface
- programmable periodic interrupt generator
  interrupt rate from micro-seconds to seconds
  highly accurate time base (5 PPM)
- 5 bit sense switch, readable by the CPU
- hardware single-step capability
- mounts directly to a standard 5 1/4" disk drive

Size 8 15/16 x 5 7/8

These *hi-speed* 68020 systems are presently working at NASA, Atomic Energy Commission, other Government Agencies as well as Universities, Business, Labs, and critical applications centers, Worldwide, where speed, math crunching and multi-user, multi-tasking UNIX C level V compatability and low cost is a *must*!

For a limited time we are offering a $400.00 trade-in on your old 68XXX SBC. Must be working properly and complete with all software, cables and documentation. Call for details.

NOTE: UniFLEX is reported to run slower than OS-9 with more than several users on line - Also call or write for information on OS-9 Version 2, soon to be available. A full 68020 OS-9, with 68881 support.

MUSTANG-020™ Software

OS-9

OS-9	$350.00
Basic09	300.00
C Compiler	400.00
Fortran 77	400.00
Microware Pascal	400.00
Omegasoft Pascal	900.00
Style-Graph	495.00
Style-Spell	195.00
Style-Merge	175.00
Style-Graph-Spell-Merge	695.00
PAT w/C source	229.00
JUST w/C source	79.95
PAT/JUST Combo	249.50
Sculptor+ (see below)	995.00
COM	125.00
Cross Assembler	50.00
Cracken w/source	100.00
Disassembler	100.00

UniFLEX

UniFLEX	$450.00
Screen Editor	150.00
Sort-Merge	200.00
BASIC/PreCompiler	300.00
C Compiler	350.00
COBOL	750.00
CMODEM w/source	100.00
TMODEM w/source	100.00
X-TALK (see Ad)	99.95
Cross Assembler	50.00
Fortran 77	650.00
Sculptor+ (see below)	995.00

Option & Expansions

8 Port expansion RS-232 495.00
(total of 20 serial ports supported)

Expansion for Motorola I/O Channel Modules $195.00

** All Expansion boards:
All expansion boards for old style cabinets will require the 101 expansion cable. Systems ordered with newer PC type cabinet do not require this cable.

101 Expansion Cable $39.95

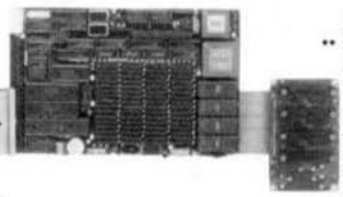
Sculptor+: We are USA distributors for Sculptor+. Call or write for site or multiple system license & discounts. OEM/Dealer.

Special for complete MUSTANG-020™ system buyers - Sculptor+ $695.00. Save $300.00!

Software Discounts

All MUSTANG-020™ system and board buyers are entitled to discounts on all listed software: 10-70% depending on item. Call or write for quotes. Discounts apply after the sale as well.

MUSTANG-020 System component prices - Effective July 1, 1986
Prices subject to change - call for latest quotes.



	MUSTANG-020 (12.50 Mhz)	$2750.00
**	Cabinet (PC or as shown)	$299.95
	5"-80 track floppy DS/DD	$269.95
	Floppy cable	$39.95
	OS-9 68K	$350.00
	Winchester cable	$39.95
	Winchester Drive 25 Mbyte	$895.00
	Xebec H/D controller	$395.00
	Shipping USA UPS	$20.00
	Total:	$5059.80

** DISCOUNT LIMITED TIME: **Complete System** **$1061.00**

Complete System :

25 Mbyte HD **$3998.80**

85 Mbyte HD **$5248.80**

OPTIONS ADD:

UniFLEX	$90.00
MC68881 I/p math processor	$275.00
16.67 Mhz MC68020	$375.00
16.67 Mhz MC68881	$375.00

This price subject to increase
Additional MUSTANG systems soon

Note: Current OS-9 (Ver. 1.2) does not address the MC68881 - Future revisions will. If the 68881 is anticipated in the future, it must be ordered with the system, when originally ordered. UniFLEX does support both the enhanced code of the 68020 and 68881 now.
OPTION BOARDS: ** *Option boards to be installed in Mustang-020 cabinets must be ordered with the extension cable.* The cabinet is too tight for direct plugon. Or specify our new PC type cabinet, with initial order.

PAT - JUST

PAT

With 'C' Source

$229.00

All OS-9 68XXX Systems

PAT FROM S. E. MEDIA -- A FULL FEATURED SCREEN ORIENTED TEXT EDITOR with all the best of PIE. For those who swore by and loved PIE, this is for **YOU!** All PIE features & much more! Too many features to list. And if you don't like ours, change or add your own. C source included. Easily configured to your CRT terminal, with special configuration section. No sweat!

68008 - 68000 - 68010 - 68020 OS-9 68K $229.00

COMBO PAT/JUST

Special $249.00

JUST

JUST from S. E. MEDIA - - Text formatter written by Ron Anderson; for dot matrix printers, provides many unique features. Output formatted to the display. User configurable for adapting to other printers. Comes set-up for Epson MX80 with Graflex. Up to 10 imbedded printer control commands. Compensates for double width printing. Includes normal line width, page numbering, margin, indent, paragraph, space, vertical skip lines, page length, centering, fill, justification, etc. Use with **PAT** or any other text editor. The ONLY stand alone text processor for the 68XXX OS-9 68K, that we have seen. And at a very **LOW PRICE!** Order from: S.E. MEDIA - see catalog this issue.

68008 - 68000 - 68010 - 68020 OS-9 68K
With 'C' source $79.95

SOFTWARE USER NOTES

A Tutorial Series

By: Ronald W. Anderson
3540 Sturbridge Court
Ann Arbor, MI 48105

From Basic Assembler to HLL's

RUSH

I am running out of both time and ideas for this column. I must have it done in a very few days because of my coming trip to Brazil, and yet, I have little in the way of inspiration, since recent days have seen me preparing for the trip rather than computing. The last column was mailed off only a few days ago. Perhaps you will forgive a little rambling rather than missing a column again.

I received a number of letters from readers with more experience with OS-9 than I, completely or partially identifying my problem with OS-9 on the Mustang 68020 system adding a CR to my LF. All those who responded indicated that the problem was with the particular OS-9 system call that was used. Those who are familiar with the "C" compiler all suggested the correct solution, to use the *write()* function rather than the *putchar()* function. My discussion of the problem and my appology to Microware for thinking bad things about the compiler were in the last column so I won't repeat them here.

I was just a little disturbed by one letter from an OS-9 user who said something like (not an exact quote) *"so you have finally moved to an operating system with some REAL capability.."* and went on to discuss how OS-9 could run rings around FLEX. First of all, who said anything about switching? I've added some basic knowledge of OS-9 to my computing repertoire because I want to use a new computer system that happens to use OS-9. I agree that OS-9 has more capabilities (by far) than FLEX in the area of multi user and multi tasking. If I needed those capabilities, I would of course use OS-9 exclusively. However, I need neither in general. Running two tasks under OS-9 or two users, greatly reduces the advantage of the 68020, which for me is its great improvement in compile time. What impresses me about the Mustang 68020 system is being able to compile a 40 page "C" program in 65 or 70 seconds, using the hard disk and ramdisk capabilities of the system, when I know that the same program would take 15 to 20 minutes to compile on a 2 MHz 6809 system.

Add a second task involving intensive hard disk accesses, and that 65 second compile time slows down by a factor of three or four, and a good deal of the advantage is lost for my applications. In addition, though I am not by any means knocking OS-9, I firmly believe that small and simple equals beauty and elegance. Though I will be using OS-9 on the 68K system for a long time to come, I certainly have no intention of "switching" from FLEX on the 6809 system that is beside me as I write this. I need no more speed than I have available when I edit text files as I am now doing, and since I am the only user of this system, why would I need multi-user capability?

My month long effort at translating PAT from PL/9 to C so I could run it on the 68020 system under OS-9 was simply to provide me with the convenience of using the same editor on both systems. Actually, when I use OS-9 in the single user mode to edit and compile software, I notice little difference externally between it and FLEX.

Speaking of small being beautiful in the context of computer software, I would like to give you some statistics. My editor PAT (pardon my using this as an example again, but it is my only good comparison point for the 6809 and the 68020), is now down to just over 16k of object code in the PL/9 version for the 6809. I did one little trick in which I used some of the program code space (that used for the code to initialize all the variables) as part of the edit buffer, since by the time I load the file to be edited, all the variables are initialized. My buffer is therefore about 29K bytes large again.

The "C" version on the 68020 system is surprisingly only about 22K bytes of code. Remember that the 68020 has longer instructions and more of them, and one would expect that the code size could double when porting or translating software packages from 6809.

To put this into perspective, I have a couple of rather complete screen editors for MS-DOS on the IBM compatibles. They are both over 100K of object code, and one of them is just about 130K! If all the software for the IBM compatibles were larger by this proportion, a 512K system would be able to do about as much as a 56K 6809 system! Those of you who run IBM compatibles know that memory gets used up quickly in those systems, and that a 10 Mbyte hard disk fills fairly quickly as well, particularly if you have some software that manipulates graphics (to be fair, the 6809 systems don't have graphics capabilities at all, of course).

I received a letter from Art Weller not long ago after sending him the chronicles of my code reduction efforts in PAT. He has just bought an IBM compatible and he was chuckling over my efforts to get PAT down to 16K, while he was already wondering if 256K on his new system would be enough to allow him to do very much with it.

Burnout?

It probably won't come as a great surprise to some of you who compute for a living and also for a hobby, that I suffer periodic burnout periods with regard to computing. Several times, I had thought that I was reaching a point where computing would soon cease to be fun for me. After a little change of activity, I've always found a new and interesting project to rekindle my enjoyment of programming again. Some people like to work crossword puzzles or play poker (or solitare), but I always say that the reason I come back to programming again and again, is that when I have finished a little or a big project, I have something to show for my time. My wife is an avid embroiderer for the same reason. I'm not a purist, however. When I decide that I simply want to relax for a while, I can kill an evening from supper time until 2 AM sitting in front of the "idiot box" being entertained. Generally when I do that, I don't want to be taught anything, nor do I want to watch anything of "cultural significance". I just want to be entertained by a good movie or two and a good TV series program.

My latest relief from computing has taken the form of having purchased a VCR. I have been recording programs when it has been inconvenient to watch them. In one case, the three networks have competing programs at the same hour. (Simon and Simon, Bill Cosby, and Our World are all on at the same hour on Thursday night here in the Detroit area). I can watch any one and record one of the others for watching an hour later when again, there is nothing to my taste to watch. I have even on occasion decided that the best of all the TV for an evening will be the 2 AM movie, and I will set up the VCR to record it so I can watch it later and enjoy fast-forwarding through the batches of commercials that come frequently on the late, late, late movies. This has provided considerable relaxation and relief from the computing burnout, as has the pursuit of a consulting job that involves assembly of a number of small electronic boxes.

So far, at least, all my burnout experiences have been totally temporary. I have always found a new and interesting project (sometimes two or three at the same time) to bring back the full enjoyment of computing that I first experienced with my little KIM-1 single board computer.

One thing that always renews my interest in computing is the availability of a new compiler for another language. I think I have a pretty complete understanding at this point of the "procedural languages" such as BASIC, Pascal, Fortran, Cobol and "C", though I don't claim to be an expert in all of them by any means. Another type of language has come to the fore recently in the context of "Expert Systems". They are called "Object Oriented" languages. An example of one of these is LISP. I can plead total ignorance of this group of languages since I have never had any of them available nor have I studied programming with them. Though I am not keenly interested in writing expert system software, I am interested in learning enough about object oriented languages to grasp what they are all about, and how they might be useful in relating information to other information. Voila! A new interest to keep me happy computing for a long time again.

Organizing a Program

In continuation of the running discussion here on programming, I'd like to present some thoughts about keeping a program easy to debug. Beginners at programming tend to write programs as one long "chunk" of code, having been encouraged by BASIC which really makes it hard to have more than a few subroutines. The best way to keep a program simple to debug is to break it down into sub functions such as the short examples presented here the last couple of times. If you have a little function to do such as to convert a hexadecimal byte to two ascii characters, you write a subroutine to do that function. The subroutine is given the byte in some standard manner (in the A accumulator or on the stack, for example) and it converts the byte to the ascii characters and "returns" with the characters in the D accumulator or on the stack. Having written such a function you write another little bit of code to test the subroutine. Send it some hex bytes and print out the results of the conversion. When you are satisfied with that subroutine, you write another to do another function in your program.

A long time ago when memory was at a premium, the main reason for writing a subroutine was to "condense" repetitive functions into a subroutine that could be "called" from several places within a program. A skilled programmer could write a subroutine with multiple entry and exit points and set some "flags" in the program to tell it how to operate on the data passed to it. Such schemes can really "telescope" code. That is, they make it possible for a routine to serve multiple purposes, and reduce the object code considerably. Maybe one of my favorite illustrations is in order here. In the old 6800 instruction set, there is only the capability to increment the X register. There is no simple instruction to add the contents of the B accumulator to X, and no way except by successive increments or storing X in memory and adding a number to it and then loading it again, to increment it by more than one. The following subroutine is entered at any of the labels, which correspond with the action. That is, to add 6 to X one does a JSR or BSR ADD6. This subroutine tucked away somewhere in a program that frequently adds 2 or 4 or 5 to X, can save much much code.

```
* SUBROUTINE TO ADD A SMALL VALUE TO X

ADD6 INX
ADD5 INX
ADD4 INX
ADD3 INX
ADD2 INX
ADD1 INX
     RTS
```

The main reason for subroutine use, however, is that of breaking the program into small chunks that can be debugged easly. Even if several functions are only needed once in the execution of the program, it is very beneficial to write them as separate subroutines and test them individually. A skilled programmer will write a 50 page program in Pascal (or Assembler) and the "main program" will consist of one page of code that does nothing but call the various procedures or subroutines, passing them the correct parameters.

In the examples of the last two columns, particularly the first of them, I discussed putting local variables on the stack. The use of local variables greatly simplifies the debug of a program. If a subroutine is allowed to modify the contents of global variables "wantonly", one can get into deep trouble with what theoreticians call "side effects". That is, a subroutine designed to do one function can accidentally modify a global variable that is being used by some other portion of the program, thus messing up the operation of the program somewhere far away from the subroutine in question. If input values are passed to a subroutine, local variables used wherever possible to hold the intermediate values calculated by the subroutine, and the final values passed back to the main program to be placed in the proper variables by that main program, one avoids a great number of mysterious bugs in a program.

This approach also of course, allows you to test each subroutine separately. When you finally execute the main program and it doesn't work, chances are that the bug is in that portion of the program. If you work out the main program first, deciding what sub functions you will need, and then implement the subroutines accordingly, you are doing what is called "Top Down Programming". If you decide what subfunctions you will need, and write and test those first, and then write the main program to call them all, you are doing what is called "Bottom Up Programming". Sometimes the process of writing the main program is just a mental exercise. You decide what functions you will need by writing the main program "in your head" and then start to code the subroutines. As you test each subroutine, perhaps you do so by adding a few lines to the main program, and by the time you have finished testing subroutines, the main program is done as well.

I usually like to tackle what I feel will be the most difficult subroutine or function first. After getting the few difficult parts of the program running, it is all down hill from that point on. I describe my approach generally as neither top down or bottom up, but "from the middle out". I suggest you use whatever works for you, and is comfortable, but whatever the order of writing the program, the technique of breaking the problem down into small pieces and solving them one at a time will produce the most results with the least effort. This technique after all, is not so different from the way we tackle other large problems in other fields than computing.

In mathematics, the process of integration is simply defining a large number of little pieces of something, and then summing some property of them, usually their area, to obtain a final result. In engineering, the same approach is used. When I was a teenager, I marveled that anyone could design something as complex as a television set. When I went to engineering school, I learned of course, that a television set consists of a tuner, an intermediate frequency amplifier, a couple of detectors, sync separators, audio amplifier, video amplifier, sweep voltage generators and amplifiers, a high voltage power supply, etc. Any one of those circuits consists of one or more "stages" each containing an active element (transistor or similar device) and a number of other components. Taken one stage at a time, the television set is not an overwhelming design problem. The same holds true in civil engineering and architecture. The big problem is considered one element at a time. It should be no surprise that a computer program is best written by breaking it into a number of smaller sub problems or sub programs.

Best of All Worlds (For Assembler Programmers)

I have just received preliminary copies of a package that consists of an editor (ED) a debugger (CRACKER) and an assembler (CRASMB). Though I mentioned the above singularly, and presently that is correct, the package will eventually contain "personality modules" for many processors.

Presently, these programs run under OS-9/68K. In my haste in preparation for a vacation, I won't be able to get into them very much until later. However, I have looked at the documentation and I have a few remarks about the package.

First of all, the first processor to have personality modules available is the 6800. The 6809 will follow shortly, and then, as I understand it, the Intel 80 series processors, and eventually the 16 bit processors. *I should mention that this package is the offering of Lloyd I/O and also sold by S.E. MEDIA.* ED is, as I have said, an editor, and the other parts of the package may be run co-resident with ED. CRACKER is a complete debugger package that allows single stepping through assembler code, disassembly, breakpoints, and the whole array of useful debugging facilities. The personality modules for CRACKER will allow it to emulate many different processors. That is, you can edit a source file for a 6800, assemble it with CRASMB, and then run and debug it with CRACKER.

This ought to be a winner package for anyone who develops assembler software for a number of different processors. Imagine writing assembling and testing software for the 6800, 6801, 6809, 6502, and Z80 as well as the newer 16 bit chips all on the same development system, and being able to download the object code to a stand alone EPROM programmer for transfer to your "target" hardware system.

Add to the above, the capability of a 68020 system, and you can actually debug code for a 6800 faster than on a 6800 system running a 6800 debugger since the 68020 will run about 40 times faster than a 1 MHz 6800 system. The same goes to a great degree for all of the 8 bit processors.

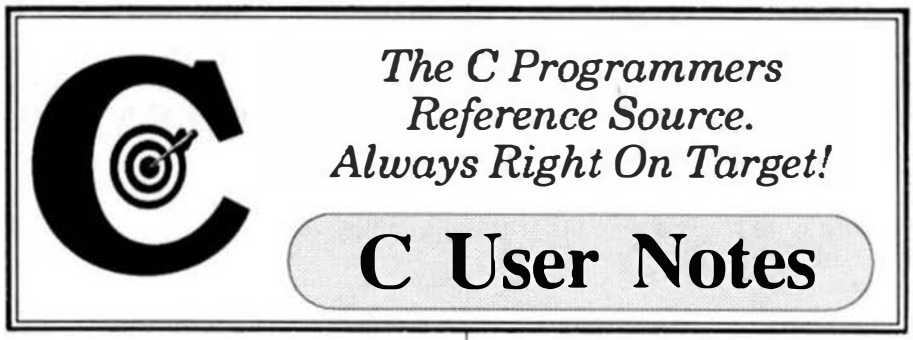
Well, though I started this column with the indication that I didn't have many ideas for topics, I see that in one evening plus half an hour the next, I have just about an average length column.

EOF

Editor's Note: I received a copy of Cracker also and it is a well done package. Knowing Frank at Lloyd I/O it's no surprise. Now, if the modules for the other devices come along promptly, it will be just what a lot of folks have called me looking for. Frank has turned out some fine software in the years that I have known him, and S.E. Media is happy to carry his line of software. If anyone knows how to write assembler/disassembler/cross-assembler software, it is certainly Frank. Cracker should be a winner!

Also, last time I was in Brazil, 43 years ago, it was certainly an 'eye opener' for a Tennessee country boy. Have fun y'all.

DMW



A Tutorial Series

By: Dr. E. M. 'Bud' Pass
1454 Latta Lane N.W.
Conyers, GA 30207
404 483-1717/4570
Computer Systems Consultants

This chapter continues the discussion of the proposed ANSI C standard and the discussion of common problem areas in the use of the C language and its libraries.

PROPOSED ANSI C STANDARD

The proposed ANSI C standard has specified the library in a manner much more precisely and completely than did K & R. For example, K & R did not specify the actions of the toupper and tolower functions on arguments not lower case and upper case, respectively.

UNIX System V (at least as of mid-1986) was used as a model, with the primary change being that several UNIX-specific functions were not included in the minimal standard library.

Definition of the following tokens is assumed in the use of the library, although they are not automatically declared in conforming C compilers (and therefore are not a part of the C language):

```
size_t  data type of "sizeof"
NULL    null pointer value
ermo    int value of error code
```

Each function in the standard library is associated with one or more header files, each intended to be made a part of any program which calls the function (with an "#include" preprocessing directive). The header files declare sets of related functions and objects, in addition to any additional types and "#define" directives to facilitate their use.

The proposed ASNI standard has specified the library in a manner much more precisely and completely than did K&R

All external functions and other identifiers in all of the header files, regardless whether they are used in a given program, are reserved. Identifiers beginning with an underscore are reserved for use by the standard library. Thus, no C programs should ever declare a function name beginning with an underscore. Although C programs are allowed to call a function with name beginning with an underscore, such use is implementation-dependent

and decreases portability. The result of passing an improper type or value of an argument to a library function is generally undefined, although many functions handle improper values in an expected manner.

The header files in the standard C library are as follows:

assert.h diagnostics
ctype.h character handling
limits.h limits and parameters
math.h mathematics
setjmp.h non-local jumps
signal.h signal handling
stdarg.h variable arguments
stdio.h input/output
stdlib.h general utilities
string.h string handling
time.h date and time

Header files may be included in any order, but must be included before the first reference to any of the functions or other objects defined in them. The user may not legally redefine any of the standard header files by providing alternate files of the same name.

Since any function may be defined in other implementations as a macro, requesting the address of a library function is non-portable. If a function is declared as a macro, each of its arguments will be evaluated exactly one time and its use will be explicitly specified with parentheses, as necessary, so that the method of definition of a library function is transparent for most purposes. If a library function which is implemented as a macro is undefined (with the use of the "#undef" directive), the function of the same name is usually uncovered.

This is important only in some cases in which it is necessary to ensure that a real function exists, such as when a function name is passed to another function. This usage is portable, since no error is generated if a macro name which has not been defined is undefined.

The header file "assert.h" is used in program diagnostics. It defines one macro (assert) and refers to one macro (NDEBUG).

The assert macro acts as if it were a function defined as follows:

```
void assert(int expr);
```

If macro name NDEBUG is defined at the point at which "assert.h" is included, assert has no effect in the program. Otherwise, it evaluates its argument and, if this evaluation produces zero, it writes information about the point of failure to the standard error file and usually calls the abort function.

The header file "ctype.h" declares the "is" functions used for classifying characters and the "to" functions used for mapping characters. The value of each argument must be representable as an unsigned char or must be equal to macro EOF.

Following are these functions:

int isalnum(int ch);
returns non-zero if ch is a letter or a decimal digit
int isalpha(int ch);
returns non-zero if ch is a letter
int iscntrl(int ch);
returns non-zero if ch is a control character
int isdigit(int ch);
returns non-zero if ch is a decimal digit
int isgraph(int ch);
returns non-zero if ch is a non-space and printable
int islower(int ch);
returns non-zero if ch is a lower-case letter
int isprint(int ch);
returns non-zero if ch is printable or space
int ispunct(int ch);
returns non-zero if ch is printable but is not a space, decimal digit, or letter
int isspace(int ch);
returns non-zero if ch is space, form feed, horizontal tab, new line, carriage return, or vertical tab

```
int isupper(int ch);
   returns non-zero if ch is
   an upper-case letter
int isxdigit(int ch);
   returns non-zero if ch is
   a hexadecimal digit
int tolower(int ch);
   returns ch if ch is not an
   upper case letter, else
   returns the lower case
   letter corresponding to ch
int toupper(int ch);
   returns ch if ch is not a
   lower case letter, else
   returns the upper case
   letter corresponding to ch
```

Note that the _tolower() and _toupper() functions have been dropped. They were the remnants of the obsolete versions of the corresponding functions which did not check their arguments before converting their case, thus returning unexpected results in some cases.

The header file "limits.h" has already been discussed. It contains a large number of macros defining implementation-defined limits and parameters of the C compiler and standard library being used.

The header file "math.h" defines several mathematical functions and several macros. For all of the functions in this group, if an input argument value is outside the legal range, the int errno is set to the macro value EDOM and the function returns a predefined value. Also, if a result is not representable as a double value, the int errno is set to the macro value ERANGE and the function returns the value of + or - the macro value HUGE_VAL on overflow or zero on underflow.

The mathematical functions in the standard C library are as follows:

```
double acos(double x);
   returns arc cosine of x
double asin(double x);
   returns arc sine of x
double atan(double x);
   returns arc tangent of x
double atan2(double y,double x);
   returns arc tangent of y/x
double cos(double x);
   returns cosine of x
double sin(double x);
   returns sine of x
double tan(double x);
   returns tangent of x
double cosh(double x);
   returns hyperbolic cosine of x
double sinh(double x);
   returns hyperbolic sine of x
double tanh(double x);
   returns hyperbolic tangent of x
double exp(double x);
   returns exponential of x
double frexp(double x, int *exp);
   returns 2 to the power x,
   normalized to [.5,1) or 0,
   and power of 2 in *exp
double ldexp(double x, int exp);
   returns x times
   (2 to the power exp)
double log(double x);
   returns natural logarithm of x
double log10(double x);
   returns base-ten logarithm of x
double modf(double x, double *ip);
   returns fractional part of x
   and integral part in *ip
double pow(double x, double y);
   returns x to power y
double sqrt(double x);
   returns square root of x
int abs(int i);
   returns absolute value of i
double ceil(double x);
   returns smallest integer not
   less than x
double fabs(double x);
   returns absolute value of x
double floor(double x);
   returns largest integer not
   greater than x
double fmod(double x, double y);
   returns remainder of x/y
```

C PROBLEM

The previous C problem involved an investigation of the implementation dependencies with respect to data type length and alignment considerations. Some typical values of these parameters are as follows:

data type	length	alignment
char	1	1
short int	2,4	1,2,4
int	2,4	1,2,4
long	4	1,2,4
float	4	1,2,4
double	6,8	1,2,4,8
char *	2,3,4	1,2,4
short int *	2,3,4	1,2,4
int *	2,4	1,2,4
long *	2,4	1,2,4
float *	2,4	1,2,4
double *	2,4	1,2,4

The number of combinations of data lengths and alignments is large, although not all combinations are encountered in current or foreseen implementations. Most implementations use eight-bit char data types and either no alignment requirements or two byte alignment on non-char data types, although some require alignment to the data type length. Moreover, some implementations have multiple pointer lengths.

If the length of a char data type is not eight bits, the lengths and alignments of the longer data types are often unusual when compared with those cases with a char data type of eight bits. Since these computers are often word-oriented, with word lengths not powers of two, the short int, int, long, and float data types are often single words and double data types are often double words, with alignment to single and sometimes double words. The char pointer length (and sometimes short int pointer length) is often longer than the other pointer lengths because of the necessity to address partial words. The value ranges which may be stored in each of the data types will almost certainly be different from the eight-bit-char case.

Regardless of the length of a char data type, the best arrangement is usually to arrange the data types from longest to shortest. This is not always the best arrangement, as alignment characteristics in a given implementation may allow shorter data types to be placed between dissimilar longer data types in spaces which would otherwise be wasted as filler bytes for alignment.

For the next C problem, consider defensive measures which might be used to help ensure portability and efficiency across implementations of compilers conforming to the proposed ANSI C standard. Also consider what minor revisions to the C language might improve the situation with respect to alignment and data type problems.

EXAMPLE C PROGRAM

Following is this month's example C program; it continues the B+ tree program started in the previous chapter.

```
insertl(a, r, h, v, z)
REF a;
int     r, *h;
ITEM  *v;
INFO *z;
{
    /* insert u to the right of a->e[r] */
  int    i, count;
 REF b;
  char    *new();

  if (a->type.leafp.k < LL)
  [ /* insert z on page *a. h = false */
     a->type.leafp.k += 1;
    *h = 0;
     for (i = a->type.leafp.k; i >= r + 2; i--)
   {
       a->type.leafp.d[i].key =
           a->type.leafp.d[i-1].key;
       a->type.leafp.d[i].count =
           a->type.leafp.d[i-1].count;
   }
     a->type.leafp.d[r+1].key = z->key;
     a->type.leafp.d[r+1].count = z->count;
}
else
 { /* page *a is full; split it and
      assign the emerging item to v */
    *h = 1;
    b = (REF)new(sizeof(*b));
   b->page_type = LEAF;
   if (r <= L)
   {
       if (r == L)
      {
          v->key = z->key;
         count = 1;
      }
      else
      {
           v->key = a->type.leafp.d[L].key;
          count = a->type.leafp.d[L].count;
          for (i = L; i >= r + 2; i--)
        {
            a->type.leafp.d[i].key =
                a->type.leafp.d[i-1].key;
            a->type.leafp.d[i].count =
                a->type.leafp.d[i-1].count;
        }
            a->type.leafp.d[r+1].key = z->key;
```

```
            a->type.leafp.d[r+1].count = z->count;
        }
          b->type.leafp.d[1].key = v->key;
          b->type.leafp.d[1].count = 1;
          for (i = 1; i <= L; i++)
        {
            b->type.leafp.d[i+1].key =
              a->type.leafp.d[i+L].key;
            b->type.leafp.d[i+1].count =
              a->type.leafp.d[i+L].count;
        }
      }
      else
        { /* insert u in right page */
          r = r - L + 1;
          v->key = a->type.leafp.d[L+1].key;
          for (i = 1; i <= r - 1; i++)
        {
            b->type.leafp.d[i].key =
              a->type.leafp.d[L+i].key;
            b->type.leafp.d[i].count =
              a->type.leafp.d[L+i].count;
        }
          b->type.leafp.d[r].key = z->key;
          b->type.leafp.d[r].count = z->count;
          for (i = r + 1; i <= L + 1; i++)
        {
            b->type.leafp.d[i].key =
              a->type.leafp.d[i+L-1].key;
            b->type.leafp.d[i].count =
              a->type.leafp.d[i+L-1].count;
        }
      }
       a->type.leafp.k = L;
       b->type.leafp.k = L + 1;
       v->p = b;
    }
}


inserti(a, r, h, v, u)
REF a;
int     r, *h;
ITEM *v, *u;
{
    /* insert u to the right of a->e[r] */
   int    i;
  REF b;
   char    *new();

   if (a->type.indexp.m < NN)
   { /* insert u on page *a. h = false */
      a->type.indexp.m += 1;
     *h = 0;
      for (i = a->type.indexp.m; i >= r + 2; i--)
     {
        a->type.indexp.e[i].key =
            a->type.indexp.e[i-1].key;
        a->type.indexp.e[i].p =
            a->type.indexp.e[i-1].p;
     }
      a->type.indexp.e[r+1].key = u->key;
      a->type.indexp.e[r+1].p = u->p;
}
else
 { /* page *a is full; split it and
    assign the emerging item to v */
    b = (REF)new(sizeof(*b));
   b->page_type = INDEX;
    if (r <= N)
   {
       if (r == N)
     {
         v->key = u->key;
        v->p = u->p;
     }
     else
     {
          v->key = a->type.indexp.e[N].key;
         v->p = a->type.indexp.e[N].p;
          for (i = N; i >= r + 2; i--)
       {
            a->type.indexp.e[i].key =
              a->type.indexp.e[i-1].key;
            a->type.indexp.e[i].p =
              a->type.indexp.e[i-1].p;
       }
          a->type.indexp.e[r+1].key = u->key;
          a->type.indexp.e[r+1].p = u->p;
     }
       for (i = 1; i <= N; i++)
     {
          b->type.indexp.e[i].key =
            a->type.indexp.e[i+N].key;
          b->type.indexp.e[i].p =
            a->type.indexp.e[i+N].p;
     }
   }
   else
     { /* insert u in right page */
       r = r - N;
        v->key = a->type.indexp.e[N+1].key;
        v->p = a->type.indexp.e[N+1].p;
        for (i = 1; i <= r - 1; i++)
     {
          b->type.indexp.e[i].key =
            a->type.indexp.e[N+i+1].key;
          b->type.indexp.e[i].p =
            a->type.indexp.e[N+i+1].p;
     }
       b->type.indexp.e[r].key = u->key;
       b->type.indexp.e[r].p = u->p;
```

```
            for (i = r + 1; i <= N; i++)
            {
                b->type.indexp.e[i].key =
                    a->type.indexp.e[i+N].key;
                b->type.indexp.e[i].p =
                    a->type.indexp.e[i+N].p;
            }
        }
        a->type.indexp.m = N;
        b->type.indexp.m = N;
        b->type.indexp.p0 = v->p;
        v->p = b;
    }
}


/* search and delete key x in b_tree a; if a page
   underflow is necessary, balance with adjacant
   page if possible, otherwise merge; h = "page a
   is undersize" */
delete(x, a, h)
int     x;
REF a;
int     *h;
{
    int     i, k, l, r;
    REF q;

    if (a->page_type == LEAF)
    {
        l = 1;
        r = a->type.leafp.k;
        while (r >= l)
          { /* binary array search */
            k = (l + r) / 2;
            if (x <= a->type.leafp.d[k].key)
              r = k - 1;
            if (x >= a->type.leafp.d[k].key)
              l = k + 1;
        };
        if (l - r > 1)
          { /* found */
            a->type.leafp.k -= 1;
            /* delete key from leaf page */
            *h = a->type.leafp.k < L;
            /* set h if underflow */
            for (i = k; i <= a->type.leafp.k; i++)
            {
                a->type.leafp.d[i].key =
                    a->type.leafp.d[i+1].key;
                a->type.leafp.d[i].count =
                    a->type.leafp.d[i+1].count;
            }
        }
        else
        {
            printf("key is not in tree\n");
            *h = 0;
        }
    }
    else
    {
        l = 1;
        r = a->type.indexp.m;
        do
            { /* binary array search */
            k = (l + r) / 2;
            if (x <= a->type.indexp.e[k].key)
              r = k - 1;
            if (x >= a->type.indexp.e[k].key)
              l = k + 1;
        }
        while (r >= l);
        if (l - r > 1)
          { /* found, now delete e[k] */
            q = a->type.indexp.e[k].p;
            delete(x, q, h);
            if (*h)
                underflow(a, q, k, h);
        }
        else
        {
            if (r == 0)
                q = a->type.indexp.p0;
            else
                q = a->type.indexp.e[r].p;
            delete(x, q, h);
            if (*h)
                underflow(a, q, r, h);
        }
    }
}
```

EOF

Basically OS-9

Dedicated to the serious OS-9 user.
The fastest growing users group world-wide!
6809 - 68020

A Tutorial Series

By: Ron Voigts
2024 Baldwin Court
Glendale Heights, IL
60139

RMB

If you are an assembly language programmer, you have certainly used RMB on many occasions. It is a handy item that can also hide details. Traditional assembly language programming has RMB reserving memory for data, hence the name Reserve Memory Bytes. In most OS-9 assembly code the effect is the same, but in a more subtle fashion.

Consider the following declarations:

```
 ORG 0
PARAM1 RMB 2
PARAM2 RMB 2
PARAM3 RMB 3
STACK RMB 200
DATASIZE EQU .
```

We reserve 3 - 2 byte areas of memory and a healthy area for the stack to reside. However, the actual memory is specified in the MOD declaration. Our MOD could be something like:

```
MOD ANEND,ANAME,$11,$81,START,DATASIZE
```

The "." above set the DATASIZE to the accumulated RMB's. Actually, a running count is kept. PARAM1 equals 0. PARAM2 equals 2. PARAM3 equals 4. STACK equals 6. And DATASIZE equal 206. The assembler uses the MOD's declaration for memory size. Using the RMB's becomes a shorthand method of assigning consecutive values. Later when writing code the parameters are referenced by their their offsets from the pointer to the start of data memory. Usually, register U is the pointer used, so to load D with the value at PARAM2, we enter:

```
LDD U,PARAM2
```

The code generated would interpret this as:

```
LDD U,2
```

So useful is this method that most items are assigned values in this manor. Many do not even represent data areas, but real values. Take the OS-9 error numbers, they start out:

```
 ORG 200

E$PTHFUL RMB 1 PATH TABLE FULL
E$BPNUM RMB 1 BAD PATH NUMBER
E$POLL RMB 1 POLLING TABLE FAULT
E$BMODE RMB 1 BAD MODE
E$DEVOVF RMB 1 DEVICE TABLE OVERFLOW
```

and so on...

None of these error numbers are areas in memory, but are real values. The RMB turns out to be a very useful way to assign consecutive values to these labels.

Using this shorthand method does require some caution. The Relocatable Macro Assembler from Microware (the same one used in their C Compiler) uses RMB's to allocate memory. With the assembler, data would be allocated with something like:

```
VSECT
   PARAM1 RMB 2
    ENDSECT
```

Here a 2 byte memory location is reserved. The programmer has no control over the memory allocation. At link time, the memory will be assigned. In the above example, 2 bytes will be reserved in the data area.

To get around this problem, there is the CSECT. A CSECT causes the labels to be assigned values. But the linker does not reserve any memory. A CSECT might appear:

```
 CSECT
 ORG 0
LABEL1 RMB 2
LABEL2 RMB 1
LABEL3 RMB 2
 ENDSECT
```

This part of code would cause LABEL1 to equal 0, LABEL2 equal 2 and LABEL3 equal 3. No memory would be allocated for these labels.

GETTING INTO OS-9

Part of my job in writing this column is to tear into OS-9. I try to learn what makes it work. Although I spend a lot of time reading the various manuals, much of my research comes out of the files in the DEFS directory. They provide a wealth of information. But sifting through all those RMB's can be a real pain. Most are 1 and 2 bytes with a few specifying 3 and 4. Going through a long list of RMB's and tallying them by hand grows old fast.

I spend a lot of time with the direct page variables used by the system. I sat down with them one night and tallied them by hand. There were 53 in all and it was a good method to learn adding in hex. My OS9DEFS file contained 505 labels. I concluded I was not going to do all of them. There had to be an easier way.

There is! My interests were in the file OS9DEFS. So, I wrote a little program called LABELS.

```
IFP1
USE .../DEFS/OS9DEFS
ENDC
END
```

This is "do nothing" program. I assembled with:

```
ASM LABELS S L M #12K >/P
```

The L tells it to list. The S means include symbols. And the M is for Motorola format (non OS-9). The result was a neat listing of all the labels with their hex values. As a bonus, they were in alphabetical order.

Now, if you want a more detailed listing try this. Copy a temporary file from the one you want. Call it TEMP, perhaps. Edit TEMP and delete all the OPT L and OPT -l lines. Add an END at the last line. And assemble it. You'll have a listing of all the labels, their values and the accompanying comments.

PUTTING IT TO USE

Once you know where everything is located, you can use your higher level languages. This month's listing is a KBASIC program, I wrote a few months back, when I did the column about the OS-9 devices and how they're attached to the system. I created this program to tabulate the devices, their drivers, buffer memory location, the device type and use count. The program draws from the first page of memory a pointer to the location of the device table. In Level I systems, the system and the user share the same 64K of memory. So everything can be PEEKed. I'll talk about Level II a little later.

Let me explain this month's program. First, we must find the device table pointer. In my system it is at $0060. This value comes from the label listings made earlier. I don't need to know where the table is located. A DPEEK of D.DEVTBL% sets I% equal to it. Now, all I do is go by 9 byte increments, getting the necessary information and print it in tabulated form.

The subroutine GTDEV gets the location of the driver, descriptor, static storage, file manager and use count. Notice the offsets from I%. These values again come from the OS9DEFS file. They are found under Device Table Format.

MODNAM gets the module's name. At an offset from the start of the module is it's name. This comes from the OS9DEFS again, under Module Field Definitions.

I think you'll find this program shows one way to use the direct page variables and the DEFS' files. Notice that I didn't use any POKEs. Although it is quite possible to alter the system, doing so indiscriminately could be extremely fatal. My best advice is to leave things as they are.

I should mention a few things about the *KBASIC*. The line "/r/kbasic, /r/ks.run" instructs the compiler that it can make temporary files in the directory KBASIC on my RAM disk. The files it uses for compiling are in the directory KS.RUN. When KBASIC runs it changes its working directory frequently. If you don't include this line it will go to its default directories.

Also, this program uses DPEEK to return 2 byte values. Many BASIC's do not have this. They return single byte values with a PEEK. If you were to translate this program into BASIC09, the line:

```
Driv%=dpeek(i%)
would become

Driv=PEEK(i)*256+PEEK(i+1)
```

I dropped the "%", since BASIC09 requires you to predeclare the variables as integer values.

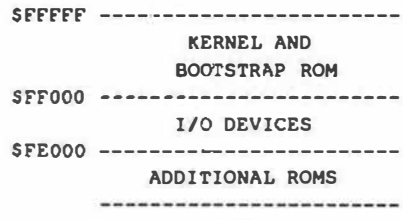
Another item worth mentioning is the reverse slashes in the PRINT USING statement. They define a string field. The numbers tucked between the slashes are not printed in the final code. I use them to better see the width of the field. The area between the slashes could have just as easily been left blank. Anything put between them will not be printed.

With the exception of TIMES, which is a pseudo string that returns the time, everything else is fairly straight forward. The constructions used here are similar to those in BASIC09. I think you should have no trouble with it.

NOW FOR THE OTHER GUY

Level II systems differ form Level I. Level II users each get a 64K chunk of memory. The system gets its own 64K too. So, attempting to PEEK into a memory location in your area will show what's is there, but it isn't the direct page variables. Level II users have to use system calls to look into other areas of memory.

It is easiest to view memory by its extended adresses. This would make available $00000 to $FFFFF. Not all systems would have memory at these locations. The amount of memory would depend on how much RAM is available. A typical memory map for the Level II system would appear:



There are a number of routines that can be used to access the different memory areas. Memory can be moved using F$MOVE. Memory can be looked at using calls like f$LDAXY or F$LDDXY. Some require the Task number, others the DAT image pointer. A simple way to get information from another area is the F$CPYMEM. This Level II call copies memory from some area to yours. Entry to the routine requires register D to contain the DAT image pointer, X the offset of the block to begin copy, Y the byte count and U the destination buffer.

A simple call to get the systems direct page would be:

```
LDD #0000 DAT IMAGE
LDX #0000 ZERO OFFSET
LDY #256 THE NUMBER OF BYTES
LDU BUFFER
OS9 F$CPYMEM
```

At the end of the call the buffer will contain 256 bytes of the system's direct page.

Armed with all this information, I am sure you are ready to attack the system. Learn what you can!

LISTING

```
*
* Name: DEVICES
* By: Ron Voigts
* Date: 9-SEP-86
* To Compile: KS DEVICES
*
* Devices will return a tabulation of all
* devices attached It will include the
* device, the manger, and the driver
* Also, the storage location and user count
* This is for a Level I system
*
 dir /r/kbasic, /r/ks.run
*
Main    PRINT
        PRINT '      LISTING OF ATTACHED DEVICES';
        PRINT ' AT ';TIME$
*     The value of D.DevTbl comes from OS9DEFS file
        D.DevTbl%=$0060
        PRINT 'DEVICE          DRIVER          ';
        PRINT 'MANAGER         STORAGE    UC  '
        PRINT '-------------------------------';
        PRINT '--------------------------------'
        i%=dpeek(D.DevTbl%)
        GOSUB GtDev
        WHILE Driv%<>$0000
            Address%=Desc%
            GOSUB ModNam
            PRINT USING '\2345678901234\ ',Nam$;
            Address%=Driv%
            GOSUB ModNam
            PRINT USING '\2345678901234\ ',Nam$;
            Address%=Fmgr%
            GOSUB ModNam
            PRINT USING '\2345678901234\ ',Nam$;
            Storage$=HEX$(Stat%)
            UC$=RIGHT$(HEX$)User'),2)
            PRINT USING '\23\    \\ ',Storage$,UC$
            i%=i%+9
            GOSUB GtDev
         ENDWHILE
         END
*
*
* Returns the device locations
* This is the format of the device table
GtDev   Driv%=dpeek(i%)
        Stat%=dpeek(i%+2)
        Desc%=dpeek(i%+4)
        Fmgr%=dpeek(i%+6)
        Usrs'=peek(i%+8)
        return
*
*
* On entry Address% points to the start of a
* module
* On exit Name$ is the the modules name!
*
ModNam Nam$=""
       w%=dpeek(Address%+4)
       Nam.Addr%=Address%+w%
       N'=peek(Nam.Addr%)
       WHILE N'>0
         Nam$=Nam$+CHR$(N')
         Nam.Addr%=Nam.Addr%+1
         N'=peek(Nam.Addr%)
      ENDWHILE
      Nam$=Nam$+CHR$(N'-128)
      RETURN
```

EOF

Layout Design In Transition

David Gross
MOTOROLA
6501 William Cannon Drive West
Austin, Texas 78735-8598

INTRODUCTION

Security would not stop him; after all, he was the design manager of the whole plant. And besides, if he was bringing in his television, two dozen donuts and an industrial size coffee pot, then he surely must have a good reason. It was superbowl weekend. His intention was as simple as it was unorthodox - to ease the weekend overtime burden of a handful of very key workers. A year prior to this, these draftsmen, or layout designers as they are called in the semiconductor industry, may have just as easily been found flipping burgers or pumping gas. In that year, they had learned some basic electronics, and some elementry drafting principles. As a result, these designers had become the critical path to the completed design of the most complex 32 bit microprocessor to date. To facilitate matters, the coffee, donuts and the final quarter of the Superbowl were a small price for a great return.

Layout Design: An Overview

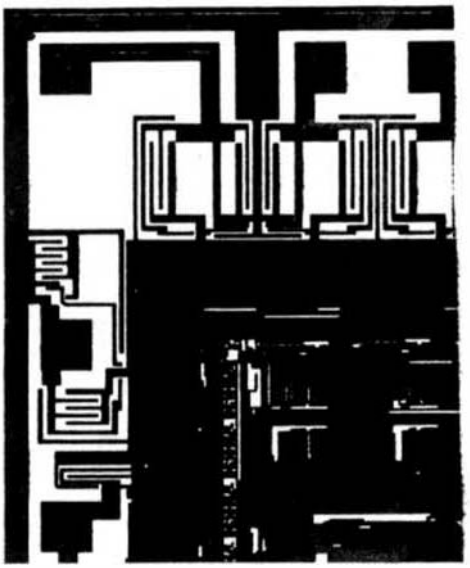
The layout design of an integrated circuit is only one part of the design cycle. The initial phase of this cycle is spent in defining what that integrated circuit will do; this takes approximately six months. Implicitly, there had been some marketing input which indicated a demand for the integrated circuit. The intended customer base may have been a specific company or the general market. Several more months are dedicated to the part specification. And approximately a half of a year will be dedicated to the logic diagram. The logic is the symbolic representation of the functions in the integrated circuit. Depending on the complexity of the part, the initial engineering work can take up to two years.

The next phase of the cycle, the layout design, is where the theoretical begins to gel into the practical. On the average the time spent in layout may be nine months to a year[1]. The final phase is the fabriation of the part, and if there are no design problems, this phase can take up to three months. The lead time is obviously a major concern especially in an industry as dynamic as the semiconductor business. Obsolescence can occur even in the design cycle.

The first phase of the actual layout of the integrated circuit is called"chip planning". This can be thought of as the roadmap of the part. The lead layout designer will interface with the design engineering team to determine the placement of "blocks" of circuitry. Here there are two basic concerns. The first concern is the electrical implications of a block's location. The other criterion is the spatial ramifications of where a block, or functional unit, will be placed. The spatial concerns are important as several integrated circuits will eventually be placed on a silicon wafer; they in turn will be diced into individual integrated circuits. The more efficiently the blocks are arranged on the individual integrated circuit, or chip, the more chips per wafer; this has direct cost saving benefits in the eventual fabrication process. The layout designer is generally more apt to be concerned with the spatial issues, while the design engineering staff will advocate the electrical issues. It is essential that there is strong two-way communication between both respective concerns. Ideally, communication is optimized because the more experienced layout designers will be placed in positions of leadership. Nevertheless, there is still great potential for problems stemming from various subjective factors such as intra-layout power struggles, inter-design power struggles, and unclear methods for conflict resolution.

Once the chip plan is reasonablly stabilized, the chip (integrated circuit) is sectioned into managable areas. These areas are championed by a section leader who may delegate small areas to be laid out, or he may layout areas himself. This layout process requires that the designer be able to convert the logic to schematic, and then he/she must convert from schematic to "drawn" form. This final conversion is performed on gridded

mylar and resembles a composite version of the shapes that will be implanted on to the wafer at different steps in the fabrication process. As an example, the metal layer will be drawn in conjunction with the contact layer, which connects the metal with other layers. Once completed, the drawing will be "digitized" which means that every layer will be hand-traced and every polygon's vertex will be transferred to a computer. Eventually, after several stages of checking, this data will be transferred to a mask making shop. Here glass representations of the layers are produced. Only clear and dark areas are on these masks. The fabrication lines will employ a photolithographic process to the eventual end, an integrated circuit.

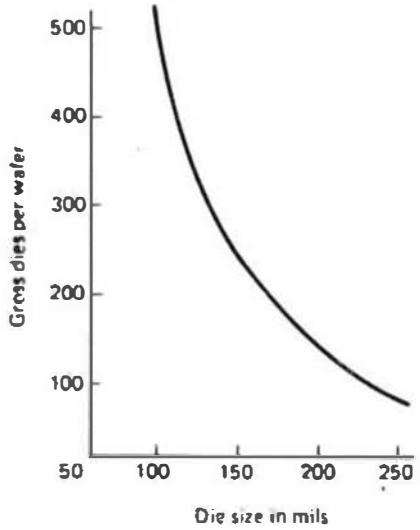


It must be noted in passing that the drawing, which is at a scale many magnitudes larger than the fabricated version, will reflect tolerances from one layer to another - or to itself. These tolerances are called "ground rules" and it is imperative that a layout designer commit these ground rules to memory. There has been a widespread movement to automate this process with the aid of computers. For example, the gridded mylar is now being replaced with computers which display a grid and enable the designer skip the tedious hand-drawing step and directly input the data into the computer.

Layout Design: What changes will the future bring?

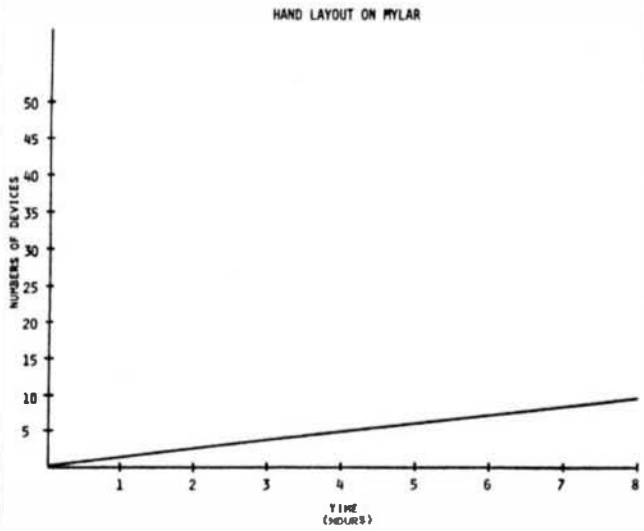
As with many highly technological occupations, that of the layout designer is constantly in flux. In the past, the lead layout designer was required to act as a liason between the design engineers and the layout team working on the chip. Members of the team were hand-picked by the manager of the layout department. The transition from a formalized layout design department, to one where the individual layout designers report directly to design engineering, is a trend that is gaining popularity. One of the key reasons for this transition is the computer. Computer aided design, *C. A. D.*, has accounted for the computer's transition from a storage device to that of an active design tool[2]. The use of a technique called the standard cell approach is one of the key reasons for the more dynamic utilization of the computer. The basic idea behind this concept is that design cycles can be greatly speeded up by modifying generic *drawings* resident on the computer as opposed to re-designing each new piece of logic from scratch. There is an implicit change in philosophy with the acceptance of this new design methodology.

One of the original benefits of hand-drawn layout was the tightly-packed drawings it yielded. The bulk of the concern has since shifted in emphasis to the point where reduced design cycles are top priority. Spacial issues, more a layout concern, take a second priority. Integrated circuits are becoming more complex, more dense, larger in transistor count, and larger in size. Microprocessors, which are in today'spersonal computers, may be a half inch on a side, or 500 mils square. There is a inverse relationship between die size and the gross die per wafer



Very large scale integration, VLSI, is becoming the standard; here 200,000 or more transistors are packed on the chip[3]. This trend toward complexity would imply a perpetuating future for the layout function. Yet, the complexity also would imply the need for specialized engineering arenas,

arenas which have less empirical underpinnings. It is doubtful that doom is on the horizion for layout designers, yet layout design, as it is presently conceived, will see dramatic shifts of emphasis in the near future. This may imply more of a supporting role for the layout design function, as opposed to an end in itself. This transition is occuring presently; it is being fueled by the gradual acceptance of automated layout techniques. Computers place standard cells, or smaller sections of these cells which are correct by construction as far as the layout tolerances are concerned. For an analysis of the increases in the per hour drawing of transistors (the smallest unit which the individual layout designer draws)



Verification and the Schedule

Before mask-making and fabrication can occur, verification of the layout must precede. When problems are encountered, the drawings must be reworked. The severity of the problem dicates the amount of rework, nevertheless, all problems impact the schedule in varying degrees. There are three types of verifications that must take place. One of these, design rule (or ground rule) verification, verifies that the spaces between materials are correct. Uncorrected, these types of problems yield direct shorts or other equally disabling situations. The second verification, is a functional verification. Functional verification implies connectivity checks and the verification of layout-to-logic. Correct layout-to-logic signifies that the layout which is drawn exactly corresponds to the logic or function the design engineer intended. This check was formerly done by hand but is now handled by computers. The layout is converted from the database format on the computer to a format whereparticular signals can be isolated through Boolean operations. These signals are also on the computer in a format called a netlist. Not only does this automated functional verification imply a reduction in the time that was formerly alloted in the layout design schedule; but it also suggests that the layout, which is verified against the logic through the netlist, can be *created* by the netlist. If this concept came to fruition, it would most certainly imply a change in the layout design function. The third check is that of performance verification. Here measurements such as resistance, capacitance and timing are verified. Presently, the layout design function demonstrates varying degrees or concern with the three types of verification. Naturally, the layout designer would be most interested in the ground rule violations he created. He, however, would be nonchalant about a circuit's timing, which is a subset of the performance verification. It must be noted that the problems that generate the most re-work are uncovered in the performance and functional verifications. Dealing with a design rule error is a trivial matter and encroaches on the design schedule very little as compared to a performance problem. For the layout design function to continue to exist given the technological advances mentioned above, the designer must metamorphose from a myopic draftsman to a quasi-engineer.

Project Management and Layout Design

The design of an integrated circuit can be realistically viewed as a project. To furthur use operations management terms, layout design can be viewed as a task. The usage of these terms are relative, but the key issue is that the problems of controlling resources is quite applicable to the design function. Likewise, some type of project management must be employed to assist in control aspects, whether it is explicit or implicit. The term "critical path" is used liberally in popular design literature; that does not imply that CPM is the de facto method for the project management of an integrated circuit's design. In fact, some of the major criticisms of CPM are very applicable in the design sphere.

There are some basic assumptions that must be understood before CPM can be sucessfully adopted as a project management tool. One of these assumptions is that projects can be identified as entities with a clear beginning and ending point. Another assumption is that a project's activity sequence can be specified and networked. Afinal assumption is that project control should focus on the critical path5.

To address the first assumption from the perspective of the design function, it is clear that the design environment is not particularly conducive to a rigidly enforced network of activities. Flexibility is not only desirable, it is a mandatory requirement. The design of an integrated circuit can be likened to a physical representation of a very complex software package. To carry the analogy furthur, if anything is coded in the software which causes a malfunction, the whole package could be rendered useless. Likewise, such an interdependent relationship exists in the designing of an integrated circuit. Partially functional integrated circuits jeapordize tenuous customer-supplier relationships. Effective project control must fluidly adapt to a changing environment. Unfortunately, this concept does not marry well with CPM as layout tasks and design tasks have a capricious nature which defy any formalized networking.

Project activity sequence may be more predictable in other arenas, but in the design cycle of an integrated circuit, this concept is unreasonable. It is, in fact, quite likley that a design engineer require that a layout designer re-designs the very first piece of logic that he gave him several months prior. It is impossible to network this into a schedule. To some extent, the use of computers in the design phase has had a "double edge sword" effect. While computers have enhanced the verification functions, they have also given designers the ability to manipulate huge quantities of data at periods that may have been formerly considered prohibitively late in the design cycle. The "networking" required in CPM would not integrate smoothly with the dynamic design requirements mentioned above.

The third assumption, that project control should focus on the critical path, is rendered equally as profitless when trying to apply tenets of CPM to the design function. The propensity for change mentioned above, was fueled by a host of inputs to the design cycle. These inputs could be marketing, circuit design, layout design or systems design (systems design inputs can be viewed as those from prototype development). At a first glance an integrated circuit's design may appear to be a series of tasks where the critical path would be readily identifiable. The interdependent and dynamic elements mentioned above makes the critical path a mirage. Change is inevitable, yet good channels of communication are mandatory if harmful effects on schedules are to be minimized. It must be noted that formalized methods of project control, such as CPM, should not be totally abandoned, just modified to account for the relentless changes.

Conclusion

Layout design is a task which is in transition. Computers, schedules, and levels of expertise are spawning a re-definition of the layout function and its spot in the design cycle. For the layout designer to adapt he/she will be required to gain familarity with other support functions. The designer will be less specialized, and have a broader perspective of the design cycle. The designer will become well-versed in the area of CAD, and must become acquainted with hardware and software from all aspects of design, not just those dedicated to the layout task. He/she will be aware of the contingent nature of the design cycle. In lieu of any formalized methodologies for project control, the layout designer must become a communicator who is as versed in the areas of electronics as he/she is in the spatial design of the circuit. By corrollary, the educational level of the future layout designer will be on the rise. The future layout design function has the potential to develop from a narrowly-defined supporting task to that of a multi-faceted contributor in an integrated design philosophy.

FOOTNOTES

1. Brian Spinks, Introduction to Integrated Circuit Layout (Englewood Cliffs, New Jersey: Prentice Hall, Inc., 1985), p. 6.

2. Martin Marshall and Larry Waller, "VLSI Pushes Super-CAD Techniques", Electronics, 31 July, 1980, pp. 73-80.

3. Spinks, p. 21.

4. M. Thomas Yin, "Layout Verification of VLSI Designs", VLSI Design, July, 1985, pp. 30-31.

5. Richard B. Chase and Nicholas J. Aquilano, Production and OperationsManagement(Homewood, Ill. : Richard D. Irwin, Inc. 1985), p. 335.

BIBLIOGRAPHY

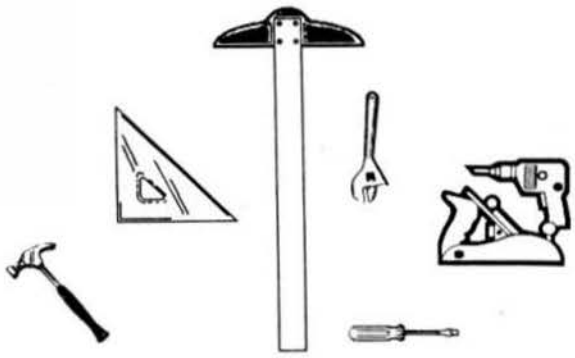
Chase, Richard B. andAquilano, Nicholas J. Production and OperationsManagement. Homewood, Ill. : Richard D. Irwin, Inc. 1985.

Marshall, Martin and Waller, Larry . "VLSI Pushes Super-CAD Techniques", Electronics, 31 July, 1980.

Spinks, Brian. Introduction to Integrated Circuit Layout. Englewood Cliffs, New Jersey: Prentice Hall, Inc., 1985.

Yin, M. Thomas. "Layout Verification of VLSI Designs", VLSI Design, July, 1985.

BASIC09 TOOLS



By & From: Ron Voigts

This is a proud moment! I felt tempted to pass out cigars and announce that it has arrived. Well I can't give you a cigar, but I can tell you about it. It is a new software package from SouthEast Media for Basic09. What makes it exciting for me is that I created it!

It is **BASIC09 TOOLS.** The TOOLS consist of 21 subroutines for Basic09. 6 were written in C Language and the remainder in assembly. All the routines are compiled down to native machine code which makes them fast and compact. It is perhaps best to show you a list of them.

1. CFILL – fills a string with characters
2. DPEEK – Double peek
3. DPOKE – Double poke
4. FPOS – Current file position
5. FSIZE – File size
6. FTRIM – removes leading spaces from a string
7. GETPR – returns the current process ID
8. GETOPT – gets 32 byte option section
9. GETUSR – gets the user ID
10. GTIME – gets the time
11. INSERT – insert a string into another
12. LOWER – converts a string into lowercase
13. READY – Checks for available input
14. SETPRIOR – changes a process priority
15. SETUSR – changes the user ID
16. SETOPT – set 32 byte option packet
17. STIME – sets the time
18. SPACE – adds spaces to a string
19. SWAP – swaps any two variables
20. SYSCALL – system call
21. UPPER – converts a string to uppercase

In writing these routines, I evaluated what was needed. I checked what was available in other basics. I also looked at other languages like Pascal and C. I examined the system calls to find out what can't be used directly from Basic09. And I looked for ways to make programming life a little easier for Basic09 programmers. The result was the 21 routines above. They handle strings, input/output, file information and access to the OS-9 system. Here is a brief description of a few of them and some examples.

Many Basics contain some variation of SWAP. There is none to be found in Basic09. It is simple and yet very handy. It allows the swapping of any two variables. The only requirement is that they be alike. Now you can write sort routines and easily SWAP records, no matter what their type.

INSERT complements the MID$ command. MID$ returns a portion of string. It is usually used in comparisons and extracting parts of a string. INSERT reverses the process. It will take some string and plunk it into another. Try this one.

```
DIM line,name:STRING[60]
DIM p:INTEGER
RUN space(line)
name:="BASIC09 TOOLS"
p:=(LEN(line)-LEN(name))/2
RUN insert(name,line,p)
```

Poof! The string 'name' will be centered in string 'line'.

Oh, the other call to SPACE fills the line with spaces. It can also be used to pad a line with spaces.

```
RUN space(name,LEN(name))
```

pads the string with spaces to its full length. There is a more generic form called CFILL.

CFILL is short for Character FILL. It repeats a string into another one. This little routine

```
DIM s:STRING[11]
RUN cfill("*",s)
```

will produce

```
******
```

The GETOPT and SETOPT routines get and replace the 32 byte option packet for an open path. Imagine you want to turn off the pause feature for some listing purpose. Normally you would use a line like:

```
SHELL "tmode .1 -pause"
```

You have the overhead of creating a new shell and executing tmode from the commands directory. Or you can get the options packet toggle the pause off and replace it. These two routines can do this for you. Another plus is when your program is ready to exit, it can return the original 32 bytes with SETOPT. Everything will be left unchanged. You can do this and a lot more. The TOOLS package has an example of the PAUSE toggling. You'll find many more uses for it.

READY is another useful routine. It returns the status of a SCF device opened on a path. In the TOOLS package is included an example of how to create a customized version of INKEY. There is an example on the disk. Imagine too! Create a program, use READY to check the keyboard and meanwhile do other things. When you get a TRUE, you know there is input available.

FPOS and FSIZE can come in real handy for file handling. They return the current position and size of a file. The returned value is a REAL number. And there is no need to rewind the file. Everything is taken care of for you.

Perhaps the most powerful one of the TOOLS is SYSCALL. This one lets you pass register information and execute any OS-9 system call. The registers are then passed back. Imagine, you can execute any system call directly from Basic09. This means that if you don't see the tool you need, you can create your own. I give one example on the disk. It shows how to create a directory from Basic09. With this one you can do almost anything the system will let you.

This is only a small sampling of the package. There is a lot more in it. All the routines generate a Basic09 Parameter Error, if you should incorrectly pass the parameters. Errors can be intercepted with ON ERROR GOTO. The routines will generate other errors as they are encountered with the exception of SYSCALL, which returns its errors as part of the B register. All the routines are under 256 bytes, they won't take up a lot of room in memory. And they are all in machine code, which means fast.

BASIC09 TOOLS is available from SOUTHEAST MEDIA. The cost is $44.95. This includes the source code as well as the tools. As time goes on I will include more examples in the BASICALLY OS-9 column. So if you're a Basic09 programmer, you'll find these TOOLS just what you need.

EOF

FORTH

A Tutorial Series

By: R. D. Lurie
9 Linda Street
Leominister, MA 01543

Most C programs require some keyboard input following a prompt. This is easy to do in C, but not so easy in FORTH. The reason is that FORTH programs originally took their input directly from a disk file or from a remote sensor, and not from the keyboard. As a result, there is no close FORTH equivalent to the C "scanf" standard I/O library function.

Prompting for keyboard input is very easy in FORTH; you only need to use the ." message" structure to display any prompt you could possible want on the output device. For the moment, I will assume that everyone knows how to do this, and not spend any more time on it.

However, taking the keyboard data is not nearly as simple, especially if you want to be compatible with the three "standard" forms of FORTH. Probably, the most general way to accomplish this is first to accept the data as a common ASCII string and, second, to parse the string in order to fit into the program's input requirements.

The INPUT$ Function

The INPUT$ function is relatively simple, but it does a lot of things. As written here, INPUT$ must be entered with an address already on the Data Stack. This is done so that the function can be as general as possible; simply enter the address (probably PAD will be the most often used) just before you call INPUT$. For example:

```
: NAME CR ." Prompt: " PAD INPUT$ ;
```

The first form of INPUT$ that I will describe in detail will work with any version of FORTH, but the second form will not work with fig-FORTH, because fig-FORTH lacks the USER variable SPAN . Since we will probably need to know the character count later, we might as well create the variable SPAN now; it does not have to be a user variable.

For all of the usual good reasons, we should also create the constant I-B-L , which stands for Input-Buffer-Length. Incidentally, you may wonder at why I choose very long and descriptive names for my definitions; it is so that there will never be a name clash with another function I may write.

We need several copies of the starting address of the string buffer, and there are several ways to make these copies. My preference is the simplest and most obvious, DUP the pointer until we have enough copies; 2 copies in this case.

I have always believed that initializing a buffer may not be absolutely necessary, but the extra time and code was well worth the cost for the peace of mind that came from the expenditure. Therefore, the next step is to fill the buffer with $20, <SP>; this uses up one copy of the address pointer. Notice that the key word is spelled BLANKS , which has a different spelling, but the same function, as BLANK in the FORTH-83 version of INPUT$.

There are 256 bytes in the buffer, but that is entirely arbitrary. I chose 256 because that is the number of bytes in a FLEX disk sector. I seriously doubt that there is any advantage to making the buffer any larger than this, but you certainly could make it smaller, if you needed to conserve the memory. As I stated before, I am

trying to write one general function which can be used in all circumstances without requiring any changes.

Since it is so often necessary to know the length of an ASCII string before it can be manipulated in FORTH, the next line of the definition counts the number of characters. The last character counted is the one which is not a <SP> at the beginning of the string of terminating <SP>'s in the buffer. In other words, all of the characters within the string are counted, including any <SP>'s, but none of the trailing <SP>'s are counted. However, initial <SP>'s are included in the count. If this is confusing, just type in the INPUT$ function, run it

with various strings, and look at the memory dump. You should then understand the action of the function; in fact, that is a good practice to follow routinely, any time.

The SWAP DROP is necessary because -TRAILING puts both the count and the address on the Data Stack, and we need to DROP the pointer to get rid of it. It is then easy to store the count in SPAN .

As you can see, the FORTH-83 version of INPUT$ does the same thing as the fig-FORTH version. It is just a shorter definition because of the presence of the SPAN user variable.

Number Entry

Entering numbers into a FORTH program in response to a prompt is almost as easy as entering an ASCII string, because we can use INPUT$ as the main part of our definition. This definition is called simply INPUT .

Since one could expect that any number entered in response to a prompt would be used immediately or stored in a convenient variable, I decided that there was no reason not to use PAD for the input buffer for numbers. Therefore, the use of INPUT is somewhat simplified over INPUT$.

Once the string is safely stored, it can be converted easily by NUMBER . The phrase PAD 1- is necessary because of the "peculiarity" of the NUMBER pointer having to be 1 less than the address of the first byte of the string to be converted. There is a good reason for this, but I will not go into it now.

Notice that there is no difference in INPUT for fig-FORTH or for FORTH-83.

I am a little unhappy with INPUT because there is no automatic error protection built into the definition. Furthermore, there is no easy way to add it later. Any sort of bad input, such as overflow, a sign in the wrong place, etc. will cause the program to crash. That is really not very good practice, but I left it so in order to make the definition a general one and not very complex.

Here is what I meant about making the definition complex by adding error protection. As the definition of INPUT now stands, it can accept input in any practical number base. But, in order to protect INPUT from overflow from too many input digits, the number of ASCII characters would have to be limited to 10 digits in decimal, 8 digits in hexadecimal, or 11 digits in octal. Now, I ask you, how can you write a simple definition to cover just those three likely possibilities?

The alternative to complexity is to leave it as is, and be

careful with your typing. I don't like that, and I would appreciate any suggestions.

There is another point of caution regarding INPUT , but it is not an error caused by incorrect data. Note must be taken of the fact that the output from INPUT is always a 32-bit number. If you are using INPUT to fetch a 16-bit number, then you must follow it with a DROP . For example, entering 32 in response to INPUT will put 0 32 on the Data Stack, and a following word expecting a 16-bit number will read the 0 and not the 32! Therefore, you must drop the 0 before you ask for the 32.

A Special Case

There are times when we only want a single digit as data, as we might if we were making a selection from a short menu. My suggestion is contained in the definition of INPUT1 .

The first thing to do is to get the keyboard input and insure that it actually is an ASCII digit. If it is, then we convert the ASCII into binary by subtracting the ASCII value for "0" from the data on the stack. The remainder must be the input number, in binary.

If the input fails the test for an ASCII digit, then it is discarded. RECURSE (also known as MYSELF in fig-FORTH) forces recursion.That is, the function is repeated as many times as necessary to get legitimate input. This also makes it impossible to enter any number of more than one digit.

Once a valid digit is entered, it is duplicated and displayed.

Notice that it is not necessary to press <RETURN> with the INPUT1 function, and the entered digit is a 16-bit integer.

For those of you without a recursion capability, there is another version which uses the BEGIN...UNTIL loop instead.

FORTH for the CoCo

A review of COLOR-FORTH from HOYT STEARNS ELECTRONICS

This version of FORTH is not new and has been reviewed before in 68 MICRO JOURNAL, but not with this viewpoint. Let me say right now that this is an excellent version of fig-FORTH for the Color Computer, but it makes extensive use of the BASIC ROMS, so it cannot be adapted to another computer, and no one should even try to do so.

COLOR-FORTH cannot make use of another operating system, so you don't need 64K, FLEX, SK*DOS, or OS-9. This should make COLOR-FORTH attractive to someone on a tight budget. You don't even need a disk system to get started. It is possible, through built-in definitions, to run strictly with tape; although, tape is in no way

as convenient as disk storage. You can even use a combination of tape and disk storage. This might be useful if you have only one disk drive.

COLOR-FORTH has several words which allow convenient coupling to the BASIC ROMS, so it is relatively easy to make use of the graphics and sound functions already there. However, there are no sound words already defined and only two very primitive graphics words. As a result, you must be prepared to write your own sound and graphics definitions. The means are there, you just have to figure them out on your own.

An "enhanced" fig-FORTH

COLOR-FORTH is a nearly exact rendition of fig-FORTH, with a few concessions made to the facts of CoCo life. The most obvious change is in the use of a 512-byte screen, instead of the conventional 1024-byte FORTH screen. Furthermore, there is a very clever, but somewhat complicated, screen editor which makes program writing about as easy as one could ask for. A disk is limited to 315 screens, or blocks, by the nature of the CoCo disk ROM. However, the number of blocks you can get on a tape is limited only by the length of the tape and your patience.

I had no trouble running a number of programs in fig-FORTH taken directly from "FORTH DIMENSIONS", including the famous and oh-so-convenient CASE structure. I found no problems with any definitions, until I tried to use some fancy manipulations of the pointer addresses CFA , LFA , NFA , etc. This may have been because the programs I was trying to use were originally written for the 8080/Z80, and I did not fully understand the algorithm the author was using. I have run into this problem with other fig-FORTHs, so I doubt that we should really be concerned. I mentioned it simply in the interest of a complete report.

COLOR-FORTH is enhanced by a large number of definitions which belong to the FORTH-79 standard. Therefore, if you keep firmly in mind that COLOR-FORTH is really a fig-FORTH, then you should find it very easy to convert programs written to the FORTH-79 standard. As nearly as I can tell,

there are only 34 essential definitions which have a different name from that used in COLOR-FORTH, or which have to be defined from scratch. The data in TABLE 1 should be all that you need to write the necessary definitions. TABLE 2 lists the needed name changes. Please let me know if I have left something out.

Conversion of FORTH-83 programs is not as easy, because of several philosophical changes made when going from FORTH-79 to FORTH-83. The most notable changes were in the definitions of PICK , ROLL , and DO...LOOP . Actually, all this really means is that you have to be more careful when you make the conversion. In rare circumstances, you may have to rewrite a DO...LOOP , but this was not necessary in the few cases that I tested.

The bad news

This review would not be complete if I did not report on those things which detract from the value of COLOR-FORTH. Mostly, they revolve around the instruction manual.

The manual does an adequate job of supplying cookbook instructions in loading COLOR-FORTH and transfering it to disk, but, beyond that, it is hardly more than a glossary of most of the defined words, some of which are simply a reprint of the FIG glossary. The tape I received for review has more definitions on it than are described in the manual; this sounds like a bonus, until you try to use the extra words! I still have not figured out a couple of them.

Another complaint about the manual is that I think that it should have more space devoted to the editor. As a general statement, I don't like screen editors as much as I do line editors. Therefore, I was really surprised at how much I liked the editor supplied with COLOR-FORTH. Unfortunately, the user is left with trying to figure out how to use the editor by fumbling around with it. This is not too bad if you have a disk, but it could be so frustrating to a tape user that he might give up in disgust before learning enough about the editor. And, without the editor, there is very little one could do with FORTH.

The last complaint I have about COLOR-FORTH is really not completely fair, in the sense that I have never seen but one FORTH which did a decent job with error messages! All fig-FORTHs that I know of are very terse and limited with explanations for errors; at times, the only error message is simply a "?". This is just not enough! Unfortunately, it is traditional, so most FORTH writers have stuck with it. I don't like it, and I say so to everybody who will listen. Oh well, at least we know where the error is, even if we don't know why.

Beginner's Problems

There is a definite trend now for language vendors to depend on other sources to teach the language that they are selling, and not to make much of any effort to include a tutorial with their own documentation. This is the case with COLOR-FORTH. Personally, I think that the trend has gone too far with everyone, not just with COLOR-FORTH. There is no way that a newcomer to FORTH could learn the language with just the information supplied

DISASSEMBLERS

SUPER SLEUTH from Computer Systems Consultants Interactive Disassembler; extremely *POWERFUL!* Disk File Binary/ASCII Examine/Change, Absolute or FULL Disassembly. XREF Generator, Label "Name Changer", and Files of "Standard Label Names" for different Operating Systems.

Color Computer SS-50 Bus (all w/ A.L. Source)
CCD (32K Req'd) Obj. Only $49.00
F, $99.00 - CCF, Obj. Only $50.00 U, $100.00
CCF, w/Source $99.00 O, $101.00
CCO, Obj. $50.00 Only
OS9 68K Obj. $100.00 w/Source $200.00

DYNAMITE+ -- Excellent standard "Batch Mode" Disassembler. Includes XREF Generator and "Standard Label" Files. Special OS-9 options w/ OS-9 Version.

CCF, Obj. Only $100.00 - CO, Obj. Only $ 59.95
F, " " $100.00 - O, object only $150.00
U, " " $300.00

PROGRAMMING LANGUAGES

PL/9 from Windrush Micro Systems -- By Graham Trott. A combination Editor Compiler Debugger. Direct source-to-object compilation delivering fast, compact, re-entrant, ROM-able, PIC. 8 & 16-bit Integers & 6-digit Real numbers for all real-world problems. Direct control over ALL System resources, including interrupts. Comprehensive library support; simple Machine Code interface; step-by-step tracer for instant debugging. 500+ page Manual with tutorial guide.

F, CCF - $198.00

PASC from S.E. Media - A Flex9 Compiler with a definite Pascal "flavor". Anyone with a bit of Pascal experience should be able to begin using PASC to good effect in short order. The PASC package comes complete with three sample programs: ED (a syntax or structure editor), EDITOR (a simple, public domain, screen editor) and CHESS (a simple chess program). The PASC package come complete with source (written in PASC) and documentation.

FLEX $95.00

WHIMSICAL from S.E. MEDIA Now supports *Real Numbers*, "Structured Programming" WITHOUT losing the Speed and Control of Assembly Language! Single-pass Compiler features unified, user-defined I/O; produces ROMable Code; Procedures and Modules (including pre-compiled Modules); many "Types" up to 32 bit Integers, 6-digit Real Numbers; unlimited sized Arrays (vectors only); Interrupt handling; long Variable Names; Variable Initialization; Include directive; Conditional compiling; direct Code insertion; control of the Stack Pointer; etc. Run-Time subroutines inserted as called during compilation. *Normally produces 10% less code than PL/9.*

F and CCF - $195.00

KANSAS CITY BASIC from S.E. Media - *Basic for Color Computer OS-9* with many new commands and sub-functions added. A full implementation of the IF-THEN-ELSE logic is included, allowing nesting to 255 levels. Strings are supported and a subset of the usual string functions such as LEFT$, RIGHT$, MID$, STRING$, etc. are included. Variables are dynamically allocated. Also included are additional features such as Peek and Poke. A must for any Color Computer user running OS-9.

CoCo OS-9 $39.95

C Compiler from Windrush Micro Systems by James McCosh. Full C for FLEX except bit-fields, including an Assembler. *Requires the TSC Relocating Assembler if user desires to implement his own Libraries.*

F and CCF - $295.00

C Compiler from Introl -- Full C except Doubles and Bit Fields, streamlined for the 6809. Reliable Compiler; FAST, efficient Code. More UNIX Compatible than most.

FLEX, CCF, OS-9 (Level II ONLY), U - $575.00

PASCAL Compiler from Lucidata -- ISO Based P-Code Compiler. Designed especially for Microcomputer Systems. Allows linkage to Assembler Code for maximum flexibility.

F and CCF 5" - $99.95 F 8" - $99.95

PASCAL Compiler from OmegaSoft (now Certified Software) -- For the *PROFESSIONAL*; ISO Based, Native Code Compiler. Primarily for Real-Time and Process Control applications. Powerful; Flexible. Requires a "Motorola Compatible" Relo. Asmb. and Linking Loader.

F and CCF - $425.00 - One Year Maint. $100.00
OS-9 68000 Version - $900.00

KBASIC - from S.E. MEDIA -- A "Native Code" BASIC Compiler which is now Fully TSC XBASIC compatible. The compiler compiles to Assembly Language Source Code. A NEW, streamlined, Assembler is now included allowing the assembly of LARGE Compiled K-BASIC Programs. Conditional assembly reduces Run-time package.

FLEX, CCF, OS-9 Compiler /Assembler $199.00

CRUNCH COBOL from S.E. MEDIA -- Supports large subset of ANSII Level I *COBOL* with many of the useful Level 2 features. Full FLEX File Structures, including Random Files and the ability to process Keyed Files. Segment and link large programs at runtime, or implemented as a set of overlays. The System requires 56K and CAN be run with a single Disk System. *A very popular product.*

FLEX, CCF, Normally $199.00
Special Introductory Price $99.95

FORTH from Stearns Electronics -- A CoCo FORTH Programming Language. Tailored to the CoCo! Supplied on Tape, transferable to disk. Written in FAST ML. Many CoCo functions (Graphics, Sound, etc.). Includes an Editor, Trace, etc. Provides CPU Carry Flag accessibility, Fast Task Multiplexing, Clean Interrupt Handling, etc. for the "Pro". Excellent "Learning" tool!

Color Computer ONLY - $58.95

!!! Please Specify Your Operating System & Disk Size !!!

Availability Legends–

F = FLEX, CCF = Color Computer FLEX
O = OS-9, CCO = Color Computer OS-9
U = UniFLEX
CCD = Color Computer Disk
CCT = Color Computer Tape

* OS-9 is a Trademark of Microware and Motorola
* FLEX is a Trademark of Technical Systems Consultants

SOUTH EAST MEDIA
5900 Cassandra Smith Rd.
Hixson, TN 37343
info (615) 842-4601
CoCo OS-9™ FLEX™
SOFTWARE

** Shipping **

Add 2% U.S.A.
(min. $2.50)
Add 5% Surface Foreign
10% Air Foreign

VISA

MasterCard

CROSS ASSEMBLERS

TRUE CROSS ASSEMBLERS from Computer Systems Consultants -- Supports 1802/5, Z-80, 6800/1/2/3/8/11/HC11, 6804, 6805/HC05/ 146805, 6809/00/01, 6502 family, 8080/5, 8020/1/2/35/C35/39/40/48/C48/49/C49/50/8748/49, 8031/51/8751, and 68000 Systems. Assembler and Listing formats same as target CPU's format. Produces machine independent Motorola S-Text. *68000 or 6809. FLEX, CCF, OS-9, UniFlex*

any object or source each - $50.00
any 3 object or source - $100.00
Set of all object $200.00 - Source $300.00

XASM Cross Assemblers for FLEX from S.E. MEDIA -- This set of 6800/1/2/3/5/8, 6301, 6502, 8080/5, and Z80 Cross Assemblers uses the familiar TSC Macro Assembler Command Line and Source Code format, Assembler options, etc., in providing code for the target CPU's.
Complete set. FLEX only - $150.00

CRASMB from LLOYD I/O -- Supports Motorola's, Intel's, Zilog's, and other's CPU syntax for these 8-Bit microprocessors: 6800, 6801, 6303, 6804, 6805, 6809, 6811 (all varieties); 6502, 1802/5, 8048 family, 8051 family, 8080/85, Z8, Z80, and TMS-7000 family. Has MACROS, Local Labels, Label X-REF, Label Length to 30 Chars. Object code formats: Motorola S-Records (text), Intel HEX-Records (text), OS9 (binary), and FLEX (binary). Written in Assembler ... e.g. Very Fast.

Availibility: MOTOROLA, INTEL, OTHER, COMPLETE SET

	CPUs	CPUs	CPUs	CPUs
FLEX9	$150	$150	$150	$399
OS9/6809	$150	$150	$150	$399
OS9/68K	------	------	------	$432

CRASMB 16.32 from LLOYD I/O -- Supports Motorola's 68000, and has same features as the 8 bit version. OS9/68K Object code Format allows this cross assembler to be used in developing your programs for OS9/68K on your OS9/6809 computer.
FLEX, CCF, OS-9/6809 $249.00

UTILITIES

NEW

Basic09 XRef from S.E. Media -- This Basic09 Cross Reference Utility is a Basic09 Program which will produce a "pretty printed" listing with each line numbered, followed by a complete cross referenced listing of all variables, external procedures, and line numbers called. Also includes a Program List Utility which outputs a fast "pretty printed" listing with line numbers. Requires Basic09 or RunB.
O & CCO obj. only -- $39.95; w/ Source - $79.95

BTree Routines - Complete set of routines to allow simple implementation of keyed files - *for your programs* - running under Basic09. A real time saver and should be a part of every serious programmers tool-box.
O & CCO obj. only - $89.95

Lucidata PASCAL UTILITIES (Requires LUCIDATA Pascal ver 3)

Availability Legends--

F = FLEX, CCF = Color Computer FLEX
O = OS-9, CCO = Color Computer OS-9
U = UniFLEX
CCO = Color Computer Disk
CCT = Color Computer Tape

*OS-9 is a Trademark of Microware and Motorola
*FLEX is a Trademark of Technical Systems Consultants

XREF -- produce a Cross Reference Listing of any text; oriented to Pascal Source.

INCLUDE -- Include other Files in a Source Text, including Binary - unlimited nesting.

PROFILER -- provides an Indented, Numbered, "Structogram" of a Pascal Source Text File; view the overall structure of large programs, program integrity, etc. Supplied in Pascal Source Code; requires compilation.
F, CCF --- EACH 5" - $40.00, 8" - $50.00

DUB from S.E. Media -- A UniFLEX BASIC decompiler Re-Create a Source Listing from UniFLEX Compiled basic Programs. Works w/ ALL Versions of 6809 UniFLEX basic.
U - $219.95

DATABASE ACCOUNTING

XDMS from Westchester Applied BusinessSystems - Powerful DBMS; M.I.. program will work on a single sided 5" disk, yet is F-A-S-T. XDMS Level I provides an "entry level" System for defining a Data Base, entering and changing the Data, and producing Reports. XDMS Level II adds the POWERFUL "GENERATE" facility with an English Language Command Structure for manipulating the Data to create new file Structures, Sort, Select, Calculate, etc. XDMS Level III adds special "Utilities" which provide additional ease in setting up a Data Base, such as copying old data into new Data Structures, changing System Parameters, etc.

XDMS System Manual - $24.95
XDMS Lvl I - F & CCF - $129.95
XDMS Lvl II - F & CCF - $199.95
XDMS Lvl III - F & CCF - $269.95

XDMS IV from Westchester Applied Business Systems - *XDMS IV* is a brand new approach to data management. It not only permits users to describe, enter and retrieve data, *but also to process entire files producing customized reports, screen displays and file output.* Processing can consist of any of a set of standard high level functions including record and field selection, sorting and aggregation, lookups in other files, special processing of record subsets, custom report formatting, totaling and subtotaling, and presentation of up to three related files as a "database" on user defined output reports.

*XDMS IV - F, CCF STAR-DOS, SK*DOS $350.00*
Upgrades to XDMS IV - $250.00

MISCELLANEOUS

TABULA RASA SPREADSHEET from Computer Systems Consultants -- TABULA RASA is similar to DESKTOP/PLAN; provides use of tabular computation schemes used for analysis of business, sales, and economic conditions. Menu-driven; extensive report-generation capabilities. Requires TSC's Extended BASIC.
F and CCF, U - $50.00, w/ Source - $100.00

DYNACALC -- Electronic Spread Sheet for the 6809 and 68000.
F, OS-9 and SPECIAL CCF - $200.00, U - $395.00
OS-9 68K - $595.00

FULL SCREEN INVENTORY/MRP from Computer Systems Consultants -- Use the Full Screen Inventory System/Materials Requirement Planning for maintaining inventories. Keeps item field file in alphabetical order for easier inquiry. Locate and/or print records matching partial or complete item, description, vendor, or attributes; find backorder or below stock levels. Print-outs in item or vendor order. MRP capability for the maintenance and analysis of Hierarchical assemblies of items in the inventory file. Requires TSC's Extended BASIC.
F and CCF, U - $50.00, w/ Source - $100.00

** Shipping **

Add 2% U.S.A.
(min. $2.50)
Add 5% Surface Foreign
10% Air Foreign

FULL SCREEN MAILING LIST from Computer Systems Consultants -- The Full Screen Mailing List System provides a means of maintaining simple mailing lists. Locate all records matching on partial or complete name, city, state, zip, or attributes for Listings or Labels, etc. Requires TSC's Extended BASIC.

F and CCF, U - $50.00. w/ Source - $100.00

DIET-TRAC Forecaster from S.E. Media -- An XBASIC program that plans a diet in terms of either calories and percentage of carbohydrates, proteins and fats (C P G%) or grams of Carbohydrate. Protein and Fat food exchanges of each of the six basic food groups (vegetable, bread, meat, skim milk, fruit and fat) for a specific individual. Sex, Age, Height, Present Weight, Frame Size, Activity Level and Basal Metabolic Rate for normal individual are taken into account. Ideal weight and sustaining calories for any weight of the above individual are calculated. Provides number of days and daily calendar after weight goal and calorie plan is determined.

F - $59.95, U - $89.95

LOW COST PROGRAM KITS from Southeast Media -- The following kits are available for FLEX on either 5 or 8 inch disk.

1. **BASIC TOOL-CHEST $29.95**
   BLISTER.CMD: pretty printer
   LINEXREF.BAS: line cross-referencer
   REMPAC.BAS, SPCPAC.BAS, COMPAC.BAS: remove superfluous code
   STRIP.BAS: superfluous line-numbers stripper
2. **FLEX UTILITIES KIT $39.95**
   CATS.CMD: alphabetically-sorted directory listing
   CATD.CMD: date-sorted directory listing
   COPYSORT.CMD: file copy, alphabetically
   COPYDATE.CMD: file copy, by date-order
   FILEDATE.CMD: change file creation date
   INFO.CMD (& INFOGMX.CMD): tells disk attributes & contents
   RELINK.CMD (& RELINK82): re-orders fragmented free chain
   RESQ.CMD: undeletes (recovers) a deleted file
   SECTORS.CMD: show sector order in free chain
   XL.CMD: super text lister
3. **ASSEMBLERS/DISASSEMBLERS UTILITIES $39.95**
   LINEFEED.CMD: 'modularise' disassembler output
   MATH.CMD: decimal, hex, binary, octal conversions & tables
   SKIP.CMD: column stripper
4. **WORD - PROCESSOR SUPPORT UTILITIES $49.95**
   FULLSTOP.CMD: checks for capitalization where required
   BSTYCIT.BAS (.BAC): Stylo to dot-matrix printer program
   NECPRINT.CMD: Stylo to dot-matrix printer filter code
5. **UTILITIES FOR INDEXING $49.95**
   MENU.BAS: selects required program from list below
   INDEX.BAC: word index
   PHRASES.BAC: phrase index
   CONTENT.BAC: table of contents
   INDXSORT.BAC: fast alphabetic sort routine
   FORMATER.BAC: produces a 2-column formatted index
   APPEND.BAC: append any number of files
   CHAR.BIN: line reader

FULL SCREEN FORMS DISPLAY from Computer Systems Consultants -- TSC Extended BASIC program supports any Serial Terminal with Cursor Control or Memory-Mapped Video Displays; substantially extends the capabilities of the Program Designer by providing a table-driven method of describing and using Full Screen Displays.

F and CCF, U - $25.00. w/ Source - $50.00

SOLVE from S.E. Media - OS-9 Levels I and II only. A Symbolic Object/Logic Verification & Examine debugger. Including inline debugging, disassemble and assemble. SOLVE IS THE MOST COMPLETE DEBUGGER we have seen for the 6809 OS-9 series! SOLVE does it all! With a rich selection of monitor, assembler, disassembler, environmental, execution and other miscellaneous commands, SOLVE is the MOST POWERFUL tool-kit item you can own! Yet, SOLVE is simple to use! With complete documentation, a snap! Everyone who has ordered this package has raved! See review - 68 Micro Journal - December 1985. No 'blind' debugging here, full screen displays, rich and complete in information presented. Since review in 68 Micro Journal, this is our fastest mover!

Levels I & II only - OS-9 Regular $149.95
SPECIAL INTRODUCTION OFFER *$69.95*

DISK UTILITIES

OS-9 VDisk from S.E. Media -- For Level I only. Use the Extended Memory capability of your SWTPC or Gimix CPU card (or similar format DAT) for FAST Program Compiles, CMD execution, high speed inter-process communications (without pipe buffers), etc. - SAVE that System Memory. Virtual Disk size is variable in 4K increments up to 960K. Some Assembly Required.

Level I OS-9 obj. $79.95; w/ Source $149.95

O-F from S.E. Media -- Written in BASIC09 (with Source), includes: REFORMAT, a BASIC09 Program that reformats a chosen amount of an OS-9 disk to FLEX Format so it can be used normally by FLEX: and FLEX, a BASIC09 Program that does the actual read or write function to the special O-F Transfer Disk; user-friendly menu driven. Read the FLEX Directory, Delete FLEX Files, Copy both directions, etc. FLEX users use the special disk just like any other FLEX disk

O - 6809/68000 $79.95

LSORT from S.E. Media - A SORT/MERGE package for OS-9 (Level I & II only). Sorts records with fixed lengths or variable lengths. Allows for either ascending or descending sort. Sorting can be done in either ASCII sequence or alternate collating sequence. Right, left or no justification of data fields available. LSORT includes a full set of comments and errors messages.

OS-9 $85.00

HIER from S.E. Media - *HIER is a modern hierarchal storage system for users under FLEX.* It answers the needs of those who have hard disk capabilities on their systems, or many files on one disk - any size. Using **HIER a regular (any) FLEX disk (8 - 5 - hard disk)** can have sub directories. By this method the problems of assigning unique names to files is less burdensome. Different files with the exact same name may be on the same disk, as long as they are in different directories. For the winchester user this becomes a must. Sub-directories are the modern day

!!! Please Specify Your Operating System & Disk Size !!!

Availability Legends--

F - FLEX, CCF - Color Computer FLEX
O - OS-9, CCO = Color Computer OS-9
U - UniFLEX
CCD - Color Computer Disk
CCT - Color Computer Tape

* OS-9 is a Trademark of Microware and Motorola
* FLEX is a Trademark of Technical Systems Consultants

** Shipping **

Add 2% U.S.A.
(min. $2.50)
Add 5% Surface Foreign
10% Air Foreign

VISA

solution that all current large systems use. **Each directory looks to FLEX like a regular file, except they have the extension '.DIR'.** A full set of directory handling programs are included, making the operation of HIER simple and straightforward. A special install package is included to install HIER to your particular version of FLEX. Some assembly required. Install indicates each byte or reference change needed. Typically - 6 byte changes in source (furnished) and one assembly of HIER is all that is required. No programming required!

** Introduction Special * $69.95*

COPYMULT from S.E. Media -- Copy LARGE Disks to several smaller disks. FLEX utilities allow the backup of ANY size disk to any SMALLER size diskettes (Hard Disk to floppies, 8" to 5", etc.) by simply inserting diskettes as requested by COPYMULT. No fooling with directory deletions, etc. COPYMULT.CMD understands normal "copy" syntax and keeps up with files copied by maintaining directories for both host and receiving disk system. Also includes BACKUP.CMD to download any size "random" type file; RESTORE.CMD to restructure copied "random" files for copying, or recopying back to the host system; and FREELINK.CMD as a "bonus" utility that "relinks" the free chain of floppy or hard disk, eliminating fragmentation.

Completely documented Assembly Language Source files included.
ALL 4 Programs (FLEX, 8" or 5") $99.50

NEW

COPYCAT from Lucidata -- *Pascal NOT required.* Allows reading TSC Mini-FLEX, SSB DOS68, and Digital Research CP/M Disks while operating under FLEX 1.0, FLEX 2.0, or FLEX 9.0 with 6800 or 6809 Systems. COPYCAT will not perform miracles, but, between the program and the manual, you stand a good chance of accomplishing a transfer. Also includes some Utilities to help out. Programs supplied in Modular Source Code (Assembly Language) to help solve unusual problems.

F and CCF 5" - $50.00 F 8" - $65.00

VIRTUAL TERMINAL from S.E. Media - Allows one terminal to do the work of several. The user may start as many as eight task on one terminal, under *VIRTUAL TERMINAL* and switch back and forth between task at will. No need to exit each one; just jump back and forth. Complete with configuration program. The best way to keep up with those background programs.

O & CCO - obj. only - $49.95

FLEX DISK UTILITIES from Computer Systems Consultants -- Eight (8) different Assembly Language (w/ Source Code) FLEX Utilities for every FLEX Users Toolbox: Copy a File with CRC Errors; Test Disk for errors; Compare two Disks; a fast Disk Backup Program; Edit Disk Sectors; Linearize Free-Chain on the Disk; print Disk Identification; and Sort and Replace the Disk Directory (in sorted order). -- PLUS -- Ten XBASIC Programs including: A BASIC Resequencer with EXTRAs over "RENUM" like check for missing label definitions, processes Disk to Disk instead of in Memory, etc. Other programs Compare, Merge, or Generate Updates between two BASIC Programs, check BASIC Sequence Numbers, compare two unsequenced files, and 5 Programs for establishing a Master Directory of several Disks, and sorting, selecting, updating, and printing paginated listings of these files. A BASIC Cross-Reference Program, written in Assembly Language, which provides an X-Ref Listing of the Variables and Reserved Words in TSC BASIC, XBASIC, and PRECOMPILER BASIC Programs.

ALL Utilities include Source≥ (either BASIC or A.L. Source Code).
F and CCF - $50.00
BASIC Utilities ONLY for UniFLEX -- $30.00

COMMUNICATIONS

CMODEM Telecommunications Program from Computer Systems Consultants, Inc. -- Menu-Driven; supports Dumb-Terminal Mode, Upload and Download in non-protocol mode, and the CP/M "Modem7" Christensen protocol mode to enable communication capabilities for almost any requirement. Written in "C".
FLEX, CCF, OS-9, UniFLEX,68000& 6809
with Source $100.00 - w/o Source $50.00

X-TALK from S.E. Media - X-TALK consists of two disks and a special cable, the hookup enables a 6809 SWTPC computer to dump UniFLEX files directly to the UniFLEX MUSTANG-020. This is the ONLY currently available method to transfer SWTPC 6809 UniFLEX files to a 68000 UniFLEX system. Gimix 6809 users may dump a 6809 UniFLEX file to a 6809 UniFLEX five inch disk and it is readable by the MUSTANG-020. The cable is specially prepared with internal connections to match the non-standard SWTPC SO/9 I/O Db25 connectors. A special SWTPC S+ cable set is also available. Users should specify which SWTPC system he/she wishes to communicate with the MUSTANG-020. The X-TALK software is furnished on two disks. One eight inch disk contains S.E. Media modem program C-MODEM (6809) and the other disk is a MUSTANG-020 five inch disk with C-MODEM (68020). Text and binary files may be directly transferred between the two systems. The C-MODEM programs are unaltered and perform as excellent modem programs also. X-TALK can be purchased with or without the special cables, but this special price is available to registered MUSTANG-020 users only.
X-TALK Complete (cable, 2 disks) $99.95
X-TALK Software (2 disks only) $69.95
X-TALK with CMODEM Source $149.95

XDATA from S.E. Media - A COMMUNICATION Package for the UniFLEX Operating System. Use with CP/M, Main Frames, other UniFLEX Systems, etc. Verifies Transmission using checksum or CRC; Re-Transmits bad blocks, etc.
U - $299.99

ASSEMBLERS

ASTRUK09 from S.E. Media -- A "Structured Assembler for the 6809" which requires the TSC Macro Assembler.
F, CCF - $99.95

Macro Assembler for TSC -- The FLEX STANDARD Assembler.
Special -- CCF $35.00; F $50.00

OSM Extended 6809 Macro Assembler from Lloyd I/O. -- Provides local labels, Motorola S-records, and Intel Hex records; XREF. GeneOrate OS-9 Memory modules under FLEX.
FLEX, CCF, OS-9 $99.00

Relocating Assembler/Linking Loader from TSC. -- Use with many of the C and Pascal Compilers.
F, CCF $150.00

MACE, by Graham Trott from Windrush Micro Systems -- Co-Resident Editor and Assembler; fast interactive A.L. Programming for small to medium-sized Programs.
F, CCF - $75.00
XMACE -- MACE w/Cross Assembler for 6800/1/2/3/8 *F, CCF - $98.00*

Availability Legends--

F = FLEX, CCF = Color Computer FLEX
O = OS-9, CCO = Color Computer OS-9
U = UniFLEX
CCD = Color Computer Disk
CCT = Color Computer Tape

* OS-9 is a Trademark of Microware and Motorola
* FLEX is a Trademark of Technical Systems Consultants

** Shipping **
Add 2% U.S.A. (min. $2.50)
Add 5% Surface Foreign
10% Air Foreign

EDITORS & WORD PROCESSING

JUST from S.E. Media -- Text Formatter developed by Ron Anderson; for Dot Matrix Printers, provides many unique features. Output "Formatted" Text to the Display. Use the FPRINT.CMD supplied for producing multiple copies of the "Formatted" Text on the Printer INCLUDING IMBEDDED PRINTER COMMANDS (very useful at other times also, and worth the price of the program by itself). "User Configurable" for adapting to other Printers (comes set up for Epson MX-80 with Graftrax); up to ten (10) imbedded "Printer Control Commands". Compensates for a "Double Width" printed line. Includes the normal line width, margin, indent, paragraph, space, vertical skip lines, page length, page numbering, centering, fill, justification, etc. Use with PAT or any other editor.

* Now supplied as a two disk set:

Disk #1: JUST2.CMD object file, JUST2.TXT PL9 source:FLEX - CC

Disk #2: JUSTSC object and source in C: FLEX - OS9 - CC

The JTSC and regular JUST C source are two separate programs. JTSC compiles to a version that expects TSC Word Processor type commands, (.pp .sp .ce etc.) Great for your older text files. The C source compiles to a standard syntax JUST.CMD object file. Using JUST syntax (.p .u .y etc.) With all JUST functions plus several additional printer formating functions. Reference the JUSTSC C source. For those wanting an excellent BUDGET PRICED word processor, with features none of the others have. This is it!

Disk (1) - PL9 FLEX only - F & CCF - $49.95
Disk Set (2) - F & CCF & OS9 (C version) - $69.95
OS-9 68K000 complete with Source - $79.95

PAT from S.E. Media - A full feature screen oriented TEXT EDITOR with all the best of "PIE™". For those who swore by and loved only PIE, this is for you! All PIE features and much more! Too many features to list. And if you don't like these, change or add your own. PL-9 source furnished. "C" source available soon. Easily configured to your CRT, with special config section.

Regular FLEX $129.50
** SPECIAL INTRODUCTION OFFER * $79.95*
SPECIAL PAT/JUST COMBO (w/source) FLEX $99.95
OS-9 68K Version $229.00
SPECIAL PAT/JUST COMBO 68K $249.00

Note: *JUST in "C" source available for OS-9*

CEDRIC from S.E. Media - A screen oriented TEXT EDITOR with availability of 'MENU' aid. Macro definitions, configurable 'permanent definable MACROS' - all standard features and the fastest 'global' functions in the west. A simple, automatic terminal config program makes this a real 'no hassel' product. Only 6K in size, leaving the average system over 165 sectors for text buffer - appx. 14,000 plus of free memory! Extra fine for programming as well as text.

Regular $129.95
SPECIAL INTRODUCTION OFFER FLEX $69.95

BAS-EDIT from S.E. Media - A TSC BASIC or XBASIC screen editor. Appended to BASIC or XBASIC, BAS-EDIT is transparent to normal BASIC/XBASIC operation. Allows editing while in BASIC/XBASIC. Supports the following functions: OVERLAY, INSERT and DUP LINE. Make editing BASIC/XBASIC programs SIMPLE! A GREAT time and effort saver. Programmers love it! NO more retyping entires lines, etc. Complete with over 25 different CRT terminal configuration overlays. *w/ Source*

FLEX, CCF, STAR-DOS Regular $69.95
Limited Special Offer: $39.95

SCREDITOR III from Windrush Micro Systems -- Powerful Screen-Oriented Editor/Word Processor. Almost 50 different commands; over 300 pages of Documentation with Tutorial. Features Multi-Column display and editing, "decimal align" columns (AND add them up automatically), multiple keystroke macros, even/odd page headers and footers, imbedded printer control codes, all justifications, "help" support, store common command series on disk, etc. Use supplied "set-ups", or remap the keyboard to your needs. Except for proportional printing, this package will DO IT ALL!

6800 or 6809 FLEX or SSB DOS, OS-9 - $175.00

SPELLB "Computer Dictionary" from S.E. Media -- OVER 150,000 words! *Look up a word from within your Editor or Word Processor (with the SPH.CMD Utility which operates in the FLEX UCS).* Or check and update the Text after entry; ADD WORDS to the Dictionary, "Flag" questionable words in the Text, "View a word in context" before changing or ignoring, etc. SPELLB first checks a "Common Word Dictionary", then the normal Dictionary, then a "Personal Word List", and finally, any "Special Word List" you may have specified. SPELLB also allows the use of Small Disk Storage systems.

F and CCF - $129.95

STYLO-GRAPH from Great Plains Computer Co. -- A full-screen oriented WORD PROCESSOR -- (uses the 51 x 24 Display Screens on CoCo FLEX/STAR-DOS, or PBJ Wordpak). Full screen display and editing; supports the Daisy Wheel proportional printers.

NEW PRICES 6809 CCF and CCO - $99.95,
F or O - $179.95, U - $299.95

STYLO-SPELL from Great Plains Computer Co. -- Fast Computer Dictionary. Complements Stylograph.

NEW PRICES 6809 CCF and CCO - $69.95,
F or O - $99.95, U - $149.95

STYLO-MERGE from Great Plains Computer Co. -- Merge Mailing List to "Form" Letters, Print multiple Files, etc., through Stylo.

NEW PRICES6809 CCF and CCO - $59.95,
F or O - $79.95, U - $129.95

STYLO-PAK -- Graph + Spell + Merge Package Deal!!!

F or O - $329.95, U - $549.95
O, 68000 $595.00

GAMES

RAPIER - 6809 Chess Program from S.E. Media -- Requires FLEX and Displays on Any Type Terminal. Features: Four levels of play. Swap side. Point scoring system. Two display boards. Change skill level. Solve Checkmate problems in 1-2-3-4 moves. Make move and swap sides. Play white or black. This is one of the strongest CHESS programs running on any microcomputer, *estimated USCF Rating 1600+ (better than most 'club' players at higher levels)*

F and CCF - $79.95

!!! Please Specify Your Operating System & Disk Size !!!

Availability Legends--

F = FLEX, CCF = Color Computer FLEX
O = OS-9, CCO = Color Computer OS-9
U = UniFLEX
CCD = Color Computer Disk
CCT = Color Computer Tape

* OS-9 is a Trademark of Microware and Motorola
* FLEX is a Trademark of Technical Systems Consultants

** Shipping **

Add 2% U.S.A.
(min. $2.50)
Add 5% Surface Foreign
10% Air Foreign

within the package. One must buy at least one book, or acquire the equivalent help somewhere else, in order to make even the most minimal use of COLOR-FORTH. Surely, some compromise can be found in order to give more help to beginners within the language package.

So, what can a beginner do to learn enough FORTH to enjoy using it? One certainly cannot learn to use FORTH without some help, and probably books are the only source open to everyone. Leo Brodie's "Starting FORTH" is usually the one recommended, but it has some problems which make it very confusing for some beginners. Unfortunately, the local book store or the library usually do not have a very good selection of FORTH books, so it is hard to examine a book before purchase. I hope to review some of the FORTH books, expecially those intended for beginners, but I can't do that here. In any case, "Starting FORTH" is OK for learning to use COLOR-FORTH.

The best alternative for the beginner is to contact the *FORTH Interest Group, P. O. Box 8231, San Jose, CA 95155*. They can supply information on a local FORTH group which would be very pleased to help the beginner get started.

```
( ********************************************************* )
( fig-FORTH version of INPUT$                               )
( ********************************************************* )

 256 CONSTANT I-B-L              ( Input-Buffer-Length       )
   0 VARIABLE SPAN               ( number of characters input)

: INPUT$ ( adr -- )
    DUP DUP                    ( make extra copies of "adr" )
    I-B-L BLANKS               ( clear buffer to all <SP>   )
    I-B-L EXPECT               ( fetch up to bfr. length of chars)
    I-B-L -TRAILING            ( count number of characters )
      SWAP DROP SPAN ! ;       ( store count & clean up Data Stk.)

( ********************************************************* )
( ********************************************************* )
( FORTH-83 version of INPUT$                                )
( ********************************************************* )

 256 CONSTANT I-B-L              ( Input-Buffer-Length       )

: INPUT$ ( adr --- )
    DUP                        ( make extra copy of "adr"   )
    256 BLANK                  ( clear buffer to all <SP>   )
    I-B-L EXPECT ;             ( fetch up to bfr. length of chars)

( ********************************************************* )
( ********************************************************* )

: INPUT  ( -- d )
    PAD INPUT$                 ( fetch the input into the buffer )
    PAD 1- NUMBER ;            ( convert the string into a number)

( ********************************************************* )
( ********************************************************* )
( Recursion required                                        )
( ********************************************************* )

: INPUT1 ( -- n )
    KEY                        ( fetch an ASCII char. from keybrd)
      DUP ?ISDIGIT             ( be sure it is a number     )
        IF                     ( it is a number             )
          ASCII 0 -            ( convert to digit by subtraction )
        ELSE                   ( not a number               )
          DROP                 ( reject invalid input       )
          RECURSE THEN ;       ( force a proper input       )

( ********************************************************* )
( ********************************************************* )
( Recursion not required                                    )
( ********************************************************* )

: INPUT1 ( -- n )
  BEGIN
    KEY                        ( fetch an ASCII char. from keybrd)
    DUP ?ISDIGIT               ( be sure it is a number     )
      IF                       ( it is a number             )
        ASCII 0 -              ( convert to digit by subtraction )
        TRUE                   ( flag to exit loop          )
      ELSE                     ( not a number               )
        DROP                   ( reject invalid input       )
        FALSE THEN             ( force a new input          )
  UNTIL ;
```

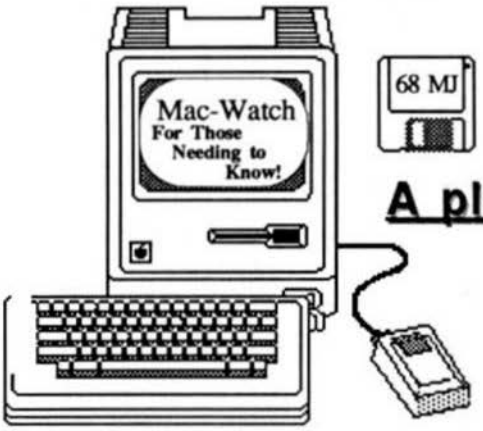
FORTH-79	COLOR-FORTH
VARIABLE	: VARIABLE 0 VARIABLE ;
2!	: 2! DUP >R ! R> 2+ ! ;
2@	: 2@ DUP >R 2+ @ R> @ ;
2CONSTANT	: 2CONSTANT <BUILDS , , DOES> 2@ ;
2OVER	: 2OVER 4 PICK 4 PICK ;
2VARIABLE	: 2VARIABLE VARIABLE 0 , ;
CONVERT	: CONVERT BEGIN 1+ DUP >R C@ BASE @ DIGIT WHILE SWAP BASE @ UM* DROP ROT BASE @ UM* D+ DPL @ 1+ IF 1 DPL +! ENDIF R> REPEAT R> ;
D-	: D- DMINUS D+ ;
D0=	: D0= OR 0= ;
D<	: D< 3 ROLL 2DUP = IF 2DROP < IF 1 ELSE 0 ENDIF ELSE > IF 2DROP 1 ELSE 2DROP 0 ENDIF ENDIF ;
D=	: D= D- D0= ;
DEPTH	: DEPTH SP@ S0 @ SWAP - 2/ ;
DMAX	: DMAX 2 PICK 2 PICK 2 PICK 2 PICK D< IF 2SWAP 2DROP ELSE 2DROP ENDIF ;
DMIN	: DMIN 2 PICK 2 PICK 2 PICK 2 PICK D< IF 2DROP ELSE 2SWAP 2DROP ENDIF ;
DU<	: DU< 2 ROLL 2DUP = IF 2DROP U< IF 1 ELSE 0 ENDIF ELSE SWAP U< IF 2DROP 1 ELSE 2DROP 0 ENDIF ENDIF
EXPECT	: EXPECT DROP QUERY ;
NOT	: NOT IF 0 ELSE 1 ENDIF ;
U.R	: U.R 0 SWAP D.R ;
U/MOD	not practical in high-level
WORD	: WORD WORD HERE ;

TABLE 1: Additions to COLOR-FORTH to permit compiling FORTH-79 (Some definitions were taken from "FF9", written by W. M. Federici).

FORTH-79	COLOR-FORTH
>IN	IN
?DUP	-DUP
ABORT"	ABORT (not exactly equivalent; no string)
BLANK	BLANKS
CMOVE	MOVE
CMOVE>	MOVE
DNEGATE	DMINUS
EXIT	;S
LAST	LATEST
NEGATE	MINUS
R@	R
THEN	ENDIF
WIPE	CLEAR
[']	'

TABLE 2: Name changes to COLOR-FORTH to permit compiling FORTH-79

EOF



The Macintosh™ Section

Reserved as a

A place for your thoughts

And ours.....

Mac-Watch

A Staff Review

M.U.D. & CheapPaint

Recently we have been receiving more and more Mac stuff to review. I guess that they are beginning to realize that because of our 68XXX following we *do* have a pretty large and loyal Macintosh following. After all, why not? With access to the many recent additions of development software for the Mac, we hit closer and closer to that target, as well as our other 68XXX readers. We welcome input directed toward any 68XXX system. And the Mac is certainly in that class.

This month we will look at several items. One a game. Yep, a game. But such a good one, so everyone around here tells me, that I have decided to tell you about it. It is called *MAZE WARS+*.

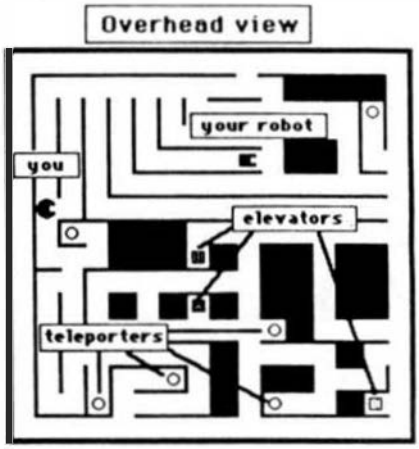
Maze Wars+ is one of the earlier arcade type games. Originally (and still is) found on many BB's in several versions. The first version was an experiment at the NASA Ames Research Center in the early 70's. Later ported to the IMLAC vector terminals at M.I.T. running on a PDP-10 with the ITS operating system. Since then Maze Wars has gone through many variations. This version is the latest and the most exciting version yet. Maze Wars+ is a commercial, souped up version. The graphics are good and the action is keen and swift.

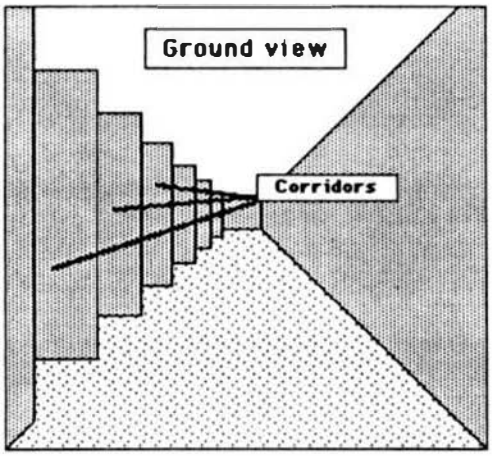
Many players play Maze Wars+ via modem, usually at 1200 or 2400 baud. In fact, on some AppleTalk networks, over 30 opponents match wits at the same time. Or if you don't want to pop for the network cost or don't have someone locally to match skills against, then play it solo against one or two interactive robots. Thats the way it is done around here. With four levels, several play options and some robot sidekicks, thing can really get going.

Maze Wars+ is so popular that it has a "*hide it from the boss*" mode. In the hide it mode you press the proper keys and it immediately changes the screen to one that seems very business like. That is if you saved a screen just for that purpose. The deal is that you use the - command-shift-3 - keys on one of your more business (or school) like screens. Then when you go into the 'hide mode' your screen displays the 'snapshot' you saved, which should always be on your disk.

Basically Maze Wars+ is a game where you and your robot sidekicks battle against opponents in a four level maze. You go in and out of halls, rooms, up and down elevators, into and out of telephone (teleporter) booths and lots of other places that can be full of surprises.

The screen has two primary views, ground view and overhead view. (see inset)





There is no way, in the space alloted that I can tell you all there is about Maze Wars+. The action is fast, exciting and above all...fun! If you like games, this one is for you. That I can tell you, for sure!

The other package we received is called M.U.D., which I assume stands for - Music, Utilities and Desk Accessories (DA). MUD was not too well documented, but sufficient for most users.

The set of programs are as follows:

1. Desk Accessories

a. Trails - this is a DA that allows the weary mind to stop and doddle on the screen. Well, not actually, the computer doddles on the screen, in a kaleidoscopic manner, what you originally doddled on the screen. You have a paint brush pointer with sizes from small to pretty large (I'm too lazy to count the screen pixels), ranging in size from 1X1 to 16X16.

A buffer menu allows various size buffers, which determine how long your doodle runs. Also, for those weary inspiration doodles, there is a repeat selection. And when you want to try another just select clear.

Combining different selections can keep your screen twirling and flashing like, gosh. I really don't know, but it is a nice *break from pounding the keys* when the noodle also seems to doodle.

b. Art Grabber+ is a pretty useful utility (DA). It greatly speed up the selecting and transfer of images between documents. No opening, selecting, closing, and then opening up your original file, to get an image onto the Clipboard. Because it is a DA you can call it anytime.

It has options to Open a MacPaint document, display the current Clipboard, grab a MacPaint image, up to the entire page depending on how much memory your system has (512K gets the whole page).

For those of you who do a lot of MacPaint projects and then import them to other applications, this is well worth the price of the entire package.

c. Cheap Paint is a lot like MacPaint (fat bits, paint brushes, pencil, eraser, text, rectangles, ovals, lines, flip-flop, paint bucket, patterns, etc.), except it is a DA. So you can use it within most applications, without having to close.

Cheap Paint has most all the MacPaint tools, plus you can have more than one project going at a time. Pages (or windows) are called Easels. You can have several easels open at the same time. They are used just like any other window.

Also, you can switch scrapbooks, even if they are on another disk. And in most applications you can leave CheapPaint open while working on your application.

Again, this one is worth the price of the whole package, especially if you are a paint fan.

2. Other programs

a. Music>Video - enables the user to compose one voice melodies with MusicWorks, and play them within VideoWorks.

b. VideoWorks Player - allows viewing a series of VideoWorks movies in sequence, without having to have a copy of VideoWorks (not supplied).

c. MusicWorks JukeBox - allows you to play through your Mac's sound system a series of music selections, without having a copy of MusicWorks.

d. Additional VideoWorks and MusicWorks selections.

All in all a fun package, with a serious side, and well worth the price. A little something for just about everyone.

Available from:

MacroMind
1028 West Wolfram
Chicago, Ill 60657
312 871-0987

E●F

Bit-Bucket

By: All of us

"Contribute Nothing - Expect Nothing", DMW '86

Mickey E. Ferguson
P. O. Box 87
Kingston Springs
Tennessee 37082

Donald M. Williams
68 Micro Journal
5900 Cassandra Smith Road
Hixson
Tennessee 37343

Dear Don,

For our phone conversation, here is the letter for the Bit Bucket. As I told you, I recently replaced the 'innards' of an old CRT terminal with Digital Research Computer's ZRT-80 terminal board. This board works fine if you happen to have an old terminal with a good monitor and keyboard and you wish to update it. The ZRT-80 will protend to be a Heath H-19, or an ADM-3A, or a couple of others. It uses a Z-80 plus a 6845 CRT controller and, I think, it is a pretty good buy at about $80. The only thing about the ZRT-80 that disappointed me was its character set. Maybe I am just spoiled by the typewriter like quality of the PC's and clones.

Knowing me as you do, you will not be surprised to find that I set about to see what could be done to improve the character set. I have learned more that I ever intended to know about character generators but have managed to duplicate the PC character set as closely as is possible within the limitations caused by the 7x9 dot character matrix used by the ZRT-80 (the PC's use an 8x8 dot matrix). I have kept the ZRT-80 (H-19) graphics characters and replaced everything else. I was extremely surprised and pleased with the end result. The readability of the screen is improved dramatically (Foxy says she can now read the it without her bi-focals, which she could not do before). In the reverse video mode, it almost equals a typewritten page for print quality. The sharpness and clarity of the character display must be seen to be appreciated!

If anyone wants one of these character generators, I am making it available on 2716 EPROM for direct replacement in the ZRT-80. While I have been talking only in terms of the ZRT-80, this character generator should work in any terminal that uses the 6845 CRT controller with a 2716 character generator and a 7x9 dot character matrix. If anyone wants this character generator, send me $15 (cash, check, or money order) and I will ship one out the next day.

I have also done an 8x16 dot matrix character generator which should equal the display quality of the PC's but I have been unable to get the ZRT-80 into the proper mode to use the 8x16 matrix. If someone can tell me how to get the ZRT-80 into the proper mode to use an 8x16 matrix, I will gladly send them a free character generator. If there is enough interest, I will put both the improved 7x9 and the 8x16 generators into a 2732 and make that available also. That way, anyone with a ZRT-80 who wants it can have display quality equal to that of the best PC's.

Since we talked on the phone, I have acquired a few 1100 A.T.& T. MAC-8 microprocessor chips. From what I have been able to find out about the MAC-8, it is the most advanced 8 bit microprocessor ever. Also, it was designed with the C language in mind. I would very much like to build a small system around one of these chips for experimentation, but unfortunately have no hardware or software data at all on the MAC-8. If any of your readers have any information on the MAC-8, I would greatly appreciate it if they would contact me IMMEDIATELY!!!!!

73's and all the best to you and Joyce.

Mickey

Greene, Johnson Inc.
15 Via Chiquita, Monterey, California 93940
(408) 375 2828

Press release: Spellswell Adds Works, More and Jazz; dictionary increased to 93,000 words.

November 10, 1986 - Monterey, California - Greene, Johnson Inc., today announced an upgrade of **Spellswell**, their popular Macintosh spell checking and proofreading program. Spellswell Version 1.3 now scans documents created with Microsoft Works, Lotus Jazz and Living Videotext **More**.

Version 1.3 includes a 93,000 word dictionary upgrade.

Spellswell also scans documents created with Apple's **MacWrite**, Microsoft Word, Living Videotext's ThinkTank, as well as any "text-only" document.

The suggested retail price of Spellswell Version 1.3 is $74.95. Greene, Johnson Inc. is offering a special introductory price of $59.95 until January 1, 1987.

According to company president Michael Greene, Spellswell Version 1.3 has the following added features:

- The dictionary has been upgraded to 93,000 words.
- It now scans documents created with Living Videotext's More, Lotus' Jazz and Microsoft Works. This is in addition to scanning MacWrite, Word, ThinkTank and Mert documents.
- Spellswell Version 1.3 has expanded proofreading capabilities, including checking for double word mistakes, like "the the".
- The speed has been greatly improved. Version 1.3 uses the Macintosh's memory more effectively to reduce the amount of disk access.
- Spellswell Version 1.3 remembers the user's "short cut" option settings.

According to reviews in MacUser, The Macintosh Journal, Nibble Mac, Semaphore Signal, InfoWorld, and many other well regarded publications, Spellswell is the highest rated spelling checker for the Macintosh.

HERBERT L. BOLOW, M.E.
39 Violet Avenue
Mineola, New York 11501

Mr. Larry Williams, Executive Editor
68 MICRO JOURNAL
P. O. Box 849
Hixson, TN 37343

Dear Larry:

The following information may be of help to some readers of the 68 Micro Journal who are also owners of Okidata Microline 83A and similar printers using half inch ribbon. It appears that my 83A printer uses only the top half of the ribbon, and thus it is possible to reverse the ribbon, and use the bottom half to get even more mileage out of them. I became aware of this only recently and do not recall reading anything about this in the 68 Micro Journal.

Another item I was not aware of until a recent trip to our local stationers, was the existence of Multi Strike Black Film ribbons. I bought one just to experiment with, and it appears to give a nice, sharp, and black impression - not as smudgy as a new nylon ribbon, as you can tell from this letter. How long they stay black I do not know yet. This depends on the quality of the film, and the coating, which appear to be quite substantial. Anyone who should like to try these can write to:

GOLDSMITH BROS.
257 JERICHO TURNPIKE
MINEOLA, NY 11501

and order OKIDATA Multi Strike Black Film ribbons, cat. #RIBL073. The cost for one ribbon is $3.75, and quantity discounts are available, namely 10% for 6 ribbons, and 20% for 12 ribbons. When threading these ribbons into your printer be sure that the shiny side faces the print head, and the dull side with the carbon coating, faces the paper.

Incidentally, Goldsmith Bros. also carry the standard nylon ribbons for the Okidata Microline 83A, as well as a selection of ribbons for other printers.

Yours truly,

MOTOROLA INC.

Semiconductor Products Group
6501 William Cannon Drive West
Austin, Texas 78735-8598

For further information contact:

READER CONTACT:
Laura Tolpan
512/440-2035

INQUIRY RESPONSE:
R.Q. Green
P.O. Box 52073
Phoenix, AZ 85072

EDITORIAL CONTACT:
Mark Vercruysse
512/928-6804

MOTOROLA ANNOUNCES A NEW MCU BULLETIN BOARD
AVAILABLE TO THE PUBLIC FREE, 24 HOURS A DAY

Austin, Texas, December 3, 1986... Motorola's Microcomputer Operation has developed an electronic bulletin board as part of their focus on customer service. The electronic bulletin board features the latest MCU literature, lead times, new products, technical updates, and custom ROM information. A Motorola sales and distributor directory is provided, along with MCU selector guides.

The bulletin board is avialable free to the public 24 hours a day, seven days a week, with any terminal and modem by dialing 512-440-2725.

FORCE COMPUTERS INC.
727 University Ave
Los Gatos, California 95030

CONTACTS:
Readers: Wayne Fischer (408) 354-3410
Editorial: Merry Shohat (408) 438-3484

CORPORATE NEWS

UniFLEX OPERATING SYSTEM NOW
SUPPORTED ON FORCE COMPUTER'S VMEbus CPU BOARDS

LOS GATOS, CA., August 28, 1986 — Force Computers, headquartered here, and Technical Systems Consultants, located in Chapel Hill, NC., have agreed to port TSC's powerful UNIX-like UniFLEX* operating system to Force's zero-wait-state VME CPU boards. Performance-hungry real-time applications will be the beneficiary of this agreement.

The agreement unites one of the most flexible real-time operating systems with the highest performance VME CPU engines based on processors in the 68000 family. It supports customers who have independently melded TSC's software and Force Computer's hardware in their own products. Cross support between Force and TSC is expected to stimulate new applications and attract new customers.

"It's an excellent arrangement where UNIX compatibility and real-time performance are mandatory," said Wayne Fischer, Force Computer's Director of Marketing. "UniFLEX is a superbly streamlined assembly language implementation of feature-rich UNIX. Ported to full 32 bit engines like Force Computer's CPU-21A, the result is a 20MHz zero wait state system that'll support true real-time and time sharing processing while offering the UNIX-style environment for applications software."

"The concept of UniFLEX — a UNIX-like command and file structure in an incredibly fast implementation — is a natural for use with Force's VMEbus engines," said Randy Lewis, Applications Manager for Force Computers. "The agreement will fine-tune the code to the hardware, making system integration a very user-friendly task." Lewis added that UniFLEX was developed strictly for processors in the 68000 family. "As speeds improve, UniFLEX will continue to take advantage of the increased performance" he said.

Features Distinguish UniFLEX

User-friendly and easy to learn, UniFLEX is a multi-user operating system supporting multiple prioritized real-time tasks, interprocessor communications, semaphores, hierarchical files, and device independent I/O. UniFLEX provides virtually all UNIX capabilities, such as multi-user timesharing and resource sharing applications. Its language support includes C, FORTRAN and BASIC compilers, interpreters and assemblers. Utilities such as editors, text processors and print spoolers are available. Also supported are third party languages including LISP, Pascal and Modula-2.

The latest version of the C compiler is optimized to the 68020 instruction set and 68881 Floating Point CoProcessor interface. The FORTRAN compiler fully supports the 68881 FPCP.

At the software development level, UniFLEX' source application code is easily portable between 68000, 68010 and 68020 processors and various VME CPU boards.

Force CPU Availability

The most powerful arrangement is UniFLEX on Force's CPU-20/21 board. This series is based on the 68020 processor and runs with no wait states at either 12.5, 16.7 or 20MHz out of 0.5 to 4MB of local memory.

In addition to the CPU-20/21 series, Force manufactures six others based on the 68000 and 68010; these boards offer a system memory range of 128K to 1MB. Four CPU boards run with no wait states, allowing maximum CPU performance.

Price and Availability

UniFLEX kernels on ROM are available immediately for use with Force's CPU-3 and CPU-20/21 series. UniFLEX for other CPUs in the Force family are available on a special order basis. A large quantity license agreement can be written with fees starting at $95/user. A single-user license is $800.

About Force Computers

The leading independent designer and manufacturer of VMEbus products, Force is now in its fifth year of operation. The company has completed 14 consecutive quarters of profitable operation. Force is headquartered in Los Gatos, California with subsidiaries in West Germany, France and the United Kingdom. Sales, service and product support are provided on a worldwide basis.

UniFLEX is a trademark of Technical Systems Consultants and UNIX is a trademark of AT&T Bell Laboratories.

Simple Winchester

by: Samuel I. Green Ph.D.

Continued From Last Month

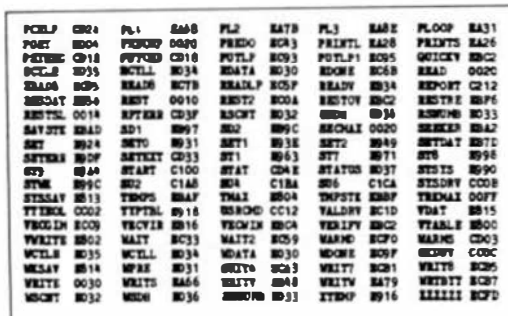
```
                          * ROUTINE TO WRITE OUT A SECTOR TO DISK

EE04 34    08       WRITDC PSHS  DP         SAVE DIRECT PAGE
>EE06 17   FFB5            LBSR  PREDO      THIS ACTS JUST ABOUT LIKE "PREAD"
EE09 86    30              LDA   #WRITE
>EE0B 17   FF1E            LBSR  COMDUT
EE0E A6    80       PUTLP  LDA   0,X+
EE10 D6    37       PUTLP1 LDB   <STATUS
EE12 C5    08              BITB  #$08       data request?
EE14 26    1C              BNE   WRTBIT     yes
EE16 C5    80              BITB  #$80       no, test busy
EE18 26    F6              BNE   PUTLP1

EE1A 30    89 FF00  WRIT3  LEAX  -256,X
EE1E 0D    37       WRIT4  TST   <STATUS
EE20 2B    FC              BMI   WRIT4
EE22 D6    37              LDB   <STATUS
EE24 C5    01              BITB  #1         ERROR?
EE26 27    04              BEQ   WRIT7
EE28 C6    10              LDB   #%00010000 MUST BE DATA FIELD
EE2A 20    04              BRA   WRIT8
EE2C C6    00       WRIT7  LDB   #0
EE2E 1C    FE              CLC
EE30 35    88       WRIT8  PULS  DP,PC

EE32 97    30       WRTBIT STA   <WDATA
EE34 20    D8              BRA   PUTLP

                           END   WARMS

0 ERROR(S) DETECTED


SYMBOL TABLE:

BELL   0007  CHECK  DE07  COMD   EDBC  COMAND ED37  COMOUT EDAC
DATE   CC0E  DREST  ED42  DRIVE  ED8C  DTMDN  ED58  DCHEK  ED6C
ERROR  ED31  FCBCK  F011  FCAIVE F00E  FINIT1 F017  FMS    D406
FORMAT 0050  FREAD  F002  FREETR F000  FTBLIF F008  FWRITE F005
GETCHR CD15  GETFIL CD2D  GETHEX CD42  HARD1  ED0E  HARD2  ED2E
HBASE  ED30  HCHECK ED4B  HDRIVE ED44  HINIT  ED54  HQDONE ED52
HRDFLG ED00  HREAD  ED01  HRESTR ED3D  HTOFIP ED30  HWRITE ED14
INBUFF CD1B  LPDEC  CD48  LUNIT  D815  MDIRV  ED8D  MOCHK  ED16
OUTCH  CD27  OPB    ED43  OCTADD CD45  OUTCH  CD0F  OUTDEC CD39
OUTHEX CD3C  PCRLF  CD24  PCMT   ED04  PRECMP 0020  PREDO  ED8B
PSTRNG CD1E  PUTCHR CD18  PUTLP  EE0E  PUTLP1 EE10  QDIDE  D812
RCTLB  ED35  RCTLL  ED34  RDATA  ED30  RDONE  ED88  READ   0020
READ1  DE00  READ6  EE00  READ8  CDF6  READC  CDCC  READLP ED04
REST   0010  REST2  CD95  RESTR1 DED9  RESTRE ED71  RESTSL 0014
RPTERR CD3F  RSCNT  ED32  RSDH   ED36  RSNDHB ED33  SECDAL 0020
SETEXT CD33  STAT   CD4E  STATUS ED37  TRKMAX CCFF  VALDRT ED98
VERIFY DED6  WAIT   EDAE  WAIT2  ED08  WARMS  CD03  WCTLB  ED35
WCTLL  ED34  WDATA  ED30  WDONE  EE16  WPRE   ED31  WRIT3  EE1E
WRIT7  EE2C  WRIT8  EE30  WRITE  0030  WRITE1 DED3  WRITDC EE04
WRTBIT EE32  WSCNT  ED32  WSDH   ED36  WSNDHB ED33


                    ***********************************************************
                    * Drivers for floppy, virtual, and winchester disk        *
                    * simultaneously resident in $E800 RAM.  Any drive of     *
                    * any type can be assigned to 1 of 4 logical flex drives  *
                    * by the SET command which is made memory resident.       *
                    *                                                         *
                    * Virtual Disk memory by Matt Bondiere 68microjournal Dec 83 *
                    *                                                         *
                    * Virtual Disk improved by Robert J. Arozeho 15 April 84  *
                    *                                                         *
                    * Winchester Hard Disk Drivers added  19 January 86       *
                    ***********************************************************

            E80C  MEMORY  SET    $E800      SWTPC memory on processor board
            CD39  OUTDEC  SET    $CD39      output decimal number
            CC12  MAXCOD  EQU    $CC12
            DED0  DOTTBL  EQU    $DED0
                  * More Variables for Drives
E800                      ORG    MEMORY
E800 E816         VTABLE  FDB    VECVIR     vector table for ref
E802 E848         VWRITE  FDB    WRITV      write routine for vector formatter
E804              TMAX    RMB    2          Maximum track number
E806 03           NDRVRS  FCB    DRVCNT     number of drivers in TYPTBL
E807 00           GDRIVE  FCB    0          global (one defined) drive
E808 00           LDRIVE  FCB    0          local drive of respective disk driver
E809 FF           RESTCT  FCB    -1         entry flag for DISKIO
E80A 00           DRIVET  FCB    0          tmp work space
E80B              DIOVEC
                  * Initial Assignments
                  * Drive Type, Local drive number (set for cold-start)
E80B 00 00                FCB    0,0        #0 floppy drive zero
E80D 01 00                FCB    1,0        #1 virtual drive zero
E80F 02 00                FCB    2,0        #2 winchester drive zero
E811 02 01                FCB    2,1        #3 winchester drive one
                  * Storage for original system and work drive
E813              SYSDRV  RMB    1
E814              WRKDRV  RMB    1
E815              VDAT    RMB    256        DAT table of virtual memory
E915              MASTRX  RMB    1
E916              STEMP   RMB    2          temporary register
                  *
                  * NOTE: ID in TYPTBL MUST NOT = 'S' OR 'W'
                  *
E918              TYPTBL
                  * Drive type 0 is floppy
E918 DE00                 FDB    VECFLP     vector for disks
E91A 03                   FCB    3          highest drive number (4 floppies)
E91B 46                   FCC    'F'        ID must be upper case
                  * Drive type 1
E91C E816                 FDB    VECVIR     vector table for virtual drive
E91E 00                   FCB    0          only one virtual drive
E91F 56                   FCC    'V'        ID is V for virtual disk
                  * Add Winchester Drive Descriptor
                  * Drive type 2
E920 EDCA                 FDB    VECWIN
E922 01                   FCB    1          two drives
E923 48                   FCC    'H'        i.d. for hard disk
            0003  DRVCNT  EQU    (*-TYPTBL)/3

                  * SET command for resetting drive type and local drive
                  * number for each of the four flex drives
                  *
                  * NOTE:  Equals, commas, & spaces are optional and ignored
                  *
                  * SET 0=F0 1=V0 2=H0 3=H1
                  * will equate flex drive 0 to floppy drive 0
                  *                        1 to virtual drive 0
                  *                        2 to winchester drive 0
                  *                        3 to winchester drive 1
                  *
                  * ? will cause a print of current status
                  * (use when preceding : in startup file if print desired)
                  * / will suppress print of current status
                  * : will suppress print if no ? present
                  * NOTE:  scan stops at first ?,/,:,<CR>
                  * S and W will assign System and work drives
                  * If S appears before a disk assignment it will set SYS drive to ALL.
                  * If S appears after a disk assignment it will that drive to SYS.
                  * Same for W
                  *
                  * ---SET S 0=V0
                  * will equate the SYS drive to ALL and equate
                  * flex drive 0 to virtual drive 0 and print
                  *
                  * ---SET 0V0S 3=W /
                  * will equate the virtual drive to flex drive 0
                  * and equate the work drive to flex drive 3
                  * with no print.
                  *
                  * ---SET will reset the processor DP register to 0 and Print.
                  *
            CD03  WARMS   SET    $CD03
            CD4F  CLSTAT  SET    $CD4F
            CD0D  INCH2   SET    $CD0D
            CD13  OUTCH2  SET    $CD13
            CD27  NXTCHR  SET    $CD27
            CD18  PUTCHR  SET    $CD18
            CD1E  PSTRNG  SET    $CD1E
            CD24  PCRLF   SET    $CD24
            CC02  TTYBOL  SET    $CC02
            CC0B  SYSDRV  SET    $CC0B
            CC0C  WRKDRV  SET    $CC0C
            CC11  LSTERM  SET    $CC11
            CC14  LBFPNT  SET    $CC14
E924              SET
E924 4F                   CLRA
E925 1F    8B             TFR    A,DP       RESET DP
E927 B6    CC11           LDA    LSTERM
E92A 81    CC02           CMPA   TTYEOL
E92D 1027  0047           LBEQ   FLEX
E931              SET0
E931 8D    7C             BSR    NEXT       GET NEXT CHR
E933 81    53             CMPA   #'S        SET SYSTEM DRIVE TO ALL
E935 26    07             BNE    SET1
E937 86    FF             LDA    #$FF
E939 B7    CC0B           STA    SYSDRV
E93C 20    F3             BRA    SET0
E93E              SET1
E93E 81    57             CMPA   #'W        SET WORK DRIVE TO ALL?
E940 26    07             BNE    SET2
E942 86    FF             LDA    #$FF
E944 B7    CC0C           STA    WRKDRV
E947 20    E8             BRA    SET0
E949              SET2
E949 80    30             SUBA   #'0        CONVERT TO BINARY
E94B 81    03             CMPA   #3         HIGHEST ALLOWED
E94D 1022  008E           LBHI   SETERR     TOO BIG
E951 B7    E80A           STA    DRIVET     SAVE IN TEMP
E954 8D    59             BSR    NEXT
E956 81    53             CMPA   #'S        SYSTEM DRIVE FLAG
E958 27    36             BEQ    SYSIS
E95A 81    57             CMPA   #'W        WORK DRIVE FLAG
E95C 27    38             BEQ    STWK
E95E 108E  E91B           LDY    #TYPTBL+3  POINT TO ALLOWED VALUES
E962 5F                   CLRB              INIT COUNTER
E963              ST1
E963 A1    A4             CMPA   ,Y         CHECK NEXT ENTRY
E965 27    0A             BEQ    ST7        MATCH
E967 31    23             LEAY   3,Y        3 ITEM / ENTRY
E969 5C                   INCB              RE-SET CNT FOR NEXT TRY
E96A F1    E806           CMPB   NDRVRS     CHECK FOR END OF TABLE
E96D 25    F4             BLO    ST1
E96F 20    6B             BRA    SETERR     NOT FOUND
E971              ST7
E971 8D    3C             BSR    NEXT
E973 80    30             SUBA   #'0        CONVERT TO BINARY
E975 A1    3F             CMPA   -1,Y       TOO BIG ?
E977 22    66             BHI    SETERR
E979 34    02             PSHS   A          SAVE LOCAL DRIVE NO.
E97B B6    E80A           LDA    DRIVET     GET GLOBAL DRIVE
E97E 48                   ASLA              *2 PER ENTRY
E97F 8E    E80B           LDX    #DIOVEC    TABLE OF DRIVER DFN'S
E982 30    86             LEAX   A,X        GET ABS ADDR
E984 35    02             PULS   A          GET DRIVER AND LOCAL DR NO.
E986 1E    89             EXG    A,B
E988 ED    84             STD    ,X         UPDATE TABLE
E98A 8D    23             BSR    NEXT       ASSIGN TO SYSTEM DRIVE ?
E98C 81    53             CMPA   #'S
E98E 26    08             BNE    ST8
E990              SYSIS
E990 B6    E80A           LDA    DRIVET
E993 B7    CC0B           STA    SYSDRV
E996 20    8C             BRA    SET
E998              ST8
E998 81    57             CMPA   #'W        ASSIGN TO WORK DRIVE ?
E99A 26    08             BNE    ST9
E99C              STWK
E99C B6    E80A           LDA    DRIVET
E99F B7    CC0C           STA    WRKDRV
E9A2 20    80             BRA    SET        CHECK FOR NEXT ENTRY
E9A4              ST9
E9A4 BE    CC14           LDX    LBFPNT     BACK UP LINE BUFFER POINTER
E9A7 30    1F             LEAX   -1,X
E9A9 BF    CC14           STX    LBFPNT
E9AC 16    FF75           LBRA   SET        CHECK FOR NEXT ENTRY
```

```
                        * GET NEXT CHR OF COMMAND LINE AND FILTER
B9AF                    NEXT
B9AF BD    CD27                 JSR    NXTCHR     NEXT CHR IN BUFFER
B9B2 81    0D                   CMPA   #$D        (CR)?
B9B4 27    70                   BEQ    PRINTS
B9B6 B1    CC03                 CMPA   TTYEOL     BREAK IN LINE?
B9B9 27    1B                   BEQ    JMPVEN
B9BB 81    2F                   CMPA   #'/        SUPPRESS PRINT
B9BD 27    17                   BEQ    JMPVEN
B9BF 81    3F                   CMPA   #'?        FLAG SUMMARY PRINT
B9C1 27    63                   BEQ    PRINTS
B9C3 81    30                   CMPA   #'0        SKIP SPACE COMMA ETC
B9C5 2D    E8                   BLT    NEXT
B9C7 81    3D                   CMPA   #'=        FILTER = TOO
B9C9 27    E4                   BEQ    NEXT
B9CB 81    61                   CMPA   #'a        LOWER CASE ?
B9CD 25    06                   BLO    EXIT
B9CF 81    7A                   CMPA   #'z        SPECIAL CHR > 'z ?
B9D1 22    02                   BHI    EXIT
B9D3 80    20                   SUBA   #'a-'A     MAKE UPPER
B9D5                    EXIT
B9D5 39                         RTS
B9D6                    JMPVEN
B9D6 32    62                   LEAS   2,S        CLEAN UP RTS ON STACK
B9D8                    FLEX
B9D8 8D    0D                   BSR    CHECK      ANY DUPLICATION ERRORS?
B9DA 26    4C                   BNE    PRINTL     YES;
B9DC 7E    CD03         FLEX    JMP    WARMS      THATS ALL FOLKS
B9DF                    ERTERR
B9DF 8E    EA99                 LDX    #MSG       ERROR MESSAGE
B9E2 BD    CD1E                 JSR    PSTRNG
B9E5 20    41                   BRA    PRINTL
                        * THIS ROUTINE CHECKS FOR DUPLICATION
B9E7                    CHECK
B9E7 7F    EB04                 CLR    ERRFLG     NOW ERROR FLAG

EA4C 84    7F                   ANDA   #$7F       STRIP ERROR BIT
EA4E 8B    30                   ADDA   #'0        CONVERT TO ASCII
EA50 BD    CD18                 JSR    PUTCHR     PRINT IT
EA53 86    0D                   LDA    #'
EA55 BD    CD18                 JSR    PUTCHR
EA58 7D    CC0B                 TST    SYSDRV     SEE IF ALL
EA5B 2B    09                   BMI    WRITS
EA5D F1    CC0B                 CMPB   SYSDRV     IS THIS THE SYS
EA60 27    04                   BEQ    WRITS
EA62 86    0D                   LDA    #'
EA64 20    02                   BRA    PL1
EA66                    WRITS
EA66 86    53                   LDA    #'S        PRINT S
EA68                    PL1
EA68 BD    CD18                 JSR    PUTCHR
EA6B 7D    CC0C                 TST    WRKDRV     SEE IF ALL
EA6E 2B    09                   BMI    WRITW
EA70 F1    CC0C                 CMPB   WRKDRV
EA73 27    04                   BEQ    WRITW
EA75 86    0D                   LDA    #'
EA77 20    02                   BRA    PL2
EA79                    WRITW
EA79 86    57                   LDA    #'W
EA7B                    PL2
EA7B BD    CD18                 JSR    PUTCHR
EA7E A6    80                   LDA    ,X+
EA80 85    80                   BITA   #$80       SEE IF ERROR
EA82 27    0A                   BEQ    PL3
EA84 34    16                   PSHS   D,X        SAVE
EA86 8E    EA99                 LDX    #MSG
EA89 BD    CD1E                 JSR    PSTRNG
EA8C 35    16                   PULS   D,X
EA8E                    PL3
EA8E BD    CD24                 JSR    PCRLF
EA91 5C                         INCB              GO TO NEXT GLOBAL DRIVE
EA92 C1    04                   CMPB   #4         END OF LIST
EA94 25    9E                   BLO    PLOOP
EA96 7E    CD03                 JMP    WARMS
                        * ERROR MESSAGE
EA99                    MSG
EA99 20 2A 2A 20                FCC    ' ** ERR **',0,4
EA9D 45 52 52 20
EAA1 2A 2A 00 04

                        * DISKIO SOURCE
                        * SELECT PROPER DRIVERS FROM SYNTHETIC DRIVE NO.
                        * ENTRY FOR DRIVE REFERENCE DRIVE ROUTINES
EAA5                    DRIVE
EAA5 34    06                   PSHS   D,DP,PC    DUMMY D,PC PLACE HOLDERS
EAA7 7C    EB09                 INC    NESTCT     KEEP TRACK OF ENTRYS FOR NESTED CALLS
EAAA 26    05                   BNE    DIO1
EAAC A6    03                   LDA    3,X        GET GLOBAL DRIVE NO.
EAAE B7    EB07                 STA    GDRIVE     SAVE IT
EAB1                    DIO1

E9EA 8E    EB0C                 LDX    #DIOVEC+1  SET TO CLR ALL ERR FLAGS
E9ED C6    04                   LDB    #4         LOOP COUNT
E9EF                    CK1
E9EF A6    84                   LDA    ,X
E9F1 84    7F                   ANDA   #$7F
E9F3 A7    84                   STA    ,X
E9F5 30    02                   LEAX   2,X
E9F7 5A                         DECB
E9F8 26    F5                   BNE    CK1
E9FA 8E    EB0B                 LDX    #DIOVEC    START AT BEGINNING OF TABLE
E9FD                    CK2
E9FD EC    81                   LDD    ,X++
E9FF 1F    12                   TFR    X,Y
EA01 8C    EB13                 CMPX   #DIOVEC+8
EA04 27    1C                   BEQ    CK8
EA06                    CK4
EA06 10A3  A4                   CMPD   ,Y
EA09 26    0D                   BNE    CK6
EA0B A6    21                   LDA    1,Y
EA0D 8A    80                   ORA    #$80       SET BOTH HI BITS
EA0F A7    21                   STA    1,Y
EA11 CA    80                   ORB    #$80
EA13 E7    1F                   STB    -1,X
EA15 7C    EB A                 INC    DRIVET     ERROR COUNT
EA18                    CK6
EA18 31    22                   LEAY   2,Y
EA1A 108C  EB13                 CMPY   #DIOVEC+8
EA1E 26    E6                   BNE    CK4
EA20 20    DB                   BRA    CK2
EA22                    CK8
EA22 7D    EB0A                 TST    DRIVET     ANY ERRORS?
EA25 39                         RTS
                        * STATUS PRINT OUT OF CURRENT DISK UPDATES
EA26                    PRINTS
EA26 32    62                   LEAS   2,S
```

```
EA28                    PRINTL
EA28 8D    BD                   BSR    CHECK      VERIFY NO DUPLICATES
EA2A 8E    EB0B                 LDX    #DIOVEC    POINTS TO DRVR'S
EA2D BD    CD24                 JSR    PCRLF
EA30 5F                         CLRB
EA31                    PLOOP
EA31 1F    98                   TFR    B,A        COPY TO A FOR PRINT
EA33 8B    30                   ADDA   #'0        CONVERT TO ASCII
EA35 BD    CD18                 JSR    PUTCHR     PRINT IT
EA38 86    3D                   LDA    #'=
EA3A BD    CD18                 JSR    PUTCHR
EA3D A6    80                   LDA    ,X+        GET DRIVER TYPE
EA3F 108E  EB1B                 LDY    #TYPTBL+3  POINT TO CHR TYPE TABLE
EA43 48                         ASLA
EA44 48                         ASLA              * FOR ITEMS / ENTRY
EA45 A6    A6                   LDA    A,Y        GET CHR TO PRINT
EA47 BD    CD18                 JSR    PUTCHR
EA4A A6    84                   LDA    ,X         GET LOCAL DRIVE NUM

EAB1 EC    66                   LDD    6,S
EAB3 ED    62                   STD    2,S        PAST RTN ADR
EAB5 AF    66                   STX    6,S        TEMP STORAGE FOR X
EAB7 CC    EAFD                 LDD    #DRVRTN    ADJUST RTN FROM DRIVES
EABA ED    64                   STD    4,S
EABC 35    06                   PULS   D          RESTORE AND FALL THRU
                        * ENTRY FOR NON-DRIVE REFERENCE DRIVE ROUTINES
EABE                    DISKIO
EABE 34    26                   PSHS   D,Y        Y TOO
EAC0 B6    EB07                 LDA    GDRIVE     GET GLOBAL DRIVE
EAC3 81    0                    CMPA   #3         TOP LIMIT FLEX
EAC5 22    46                   BHI    ERROR
EAC7 48                         ASLA              #2 BYTES / ENTRY
EAC8 108E  EB0B                 LDY    #DIOVEC    DRIVE TABLE
EACC EC    A6                   LDD    A,Y        DRV TYPE & LOGICAL DRIVE
EACE B1    EB00                 CMPA   NDRVRS     # OF DRIVE TYPES
EAD1 22    3A                   BHI    ERROR      EXCEEDED TABLE
EAD3 5D                         TSTB              CHECK FOR NON EXISTANT DRIVE
EAD4 2B    37                   BMI    ERROR
EAD6 F7    EB08                 STB    LDRIVE     UPDATE LOCAL DRIVE
EAD9 7D    EB09                 TST    NESTCT     DRIVE # ACCESS
EADC 26    02                   BNE    DIO2       NO - SKIP
EADE E7    03                   STB    3,X        SET LOCAL DRIVE # IN FCB FOR DRIVES
EAE0                    DIO2
EAE0 48                         ASLA              *2
EAE1 48                         ASLA              *4 ITEMS / ENTRY
EAE2 108E  EB16                 LDY    #TYPTBL    POINT TO VECTOR TABLE
EAE6 31    A6                   LEAY   A,Y        POINT Y TO DRIVE VECTOR
EAE8 B6    EB08                 LDA    LDRIVE     RECOVER FOR CHECK
EAEB A1    22                   CMPA   2,Y
EAED 22    1E                   BHI    ERROR
EAEF 10AE  A4                   LDY    ,Y         GET ACTUAL ADDR OF VECTOR
EAF2 E6    65                   LDB    5,S        RTN ADDR
EAF4 C0    03                   SUBB   #3         CORRECT FOR OFFSET
EAF6 31    A5                   LEAY   B,Y        ADD TO Y FOR JUMP ADDR
EAF8 10AF  64                   STY    4,S        JUMP VECTOR ON STACK
EAFB 35    A6                   PULS   D,Y,PC     RESTORE AND JUMP
                        * RETURN FROM DRIVE REF DISK ROUTINES
                        * (NON-DRIVE REFERENCE ROUTINES RETURN DIRECTLY)
EAFD                    DRVRTN
EAFD 35    10                   PULS   X          RESTORE X
EAFF 34    01                   PSHS   CC         SAVE ERROR FLAG
EB01 7A    EB09                 DEC    NESTCT     DEC NEST COUNT
EB04 2A    05                   BPL    DIO4       BRA IF NOT RETURNING TO FLEX
EB06 B6    EB07                 LDA    GDRIVE     RESTORE GLOBAL
EB09 A7    03                   STA    3,X        DRIVE IN FCB
EB0B                    DIO4
EB0B 35    81                   PULS   CC,PC      BACK TO FLEX
                        * BAD CALL EXIT
EB0D                    ERROR
EB0D 35    26                   PULS   D,Y        RESTORE
EB0F 32    62                   LEAS   2,S        CLEAN UP RTS FROM ENTRY
EB11 C6    0F                   LDB    #$0F       NO SUCH DRIVE
EB13 1A    01                   SEC               SET ERROR FLAG
EB15 39                         RTS               DONE
                        *
                        * VIRTUAL DISK DRIVERS
                        *
                        * fixed by  Robert J. Arnano
                        *
               DFD0     LRARAM  EQU    $DFD0
               FFF0     DATRAM  EQU    $FFF0
                        * VIRTUAL DISK VECTOR TABLE
EB16                    VECVD
>EB16 16   001B                 LBRA   READV
>EB19 16   002C                 LBRA   WRITV
EB1C 16    00A3                 LBRA   VERIFV
EB1F 16    00A0                 LBRA   RESTOV
EB22 16    009D                 LBRA   DRIVEV
EB25 16    009A                 LBRA   CKRDYV
EB28 16    0097                 LBRA   QUICKV
EB2B 16    0094                 LBRA   DINITV
EB2E 16    006E                 LBRA   WARMV
EB31 16    006E                 LBRA   DSEEKV
                        * FORMAT IS 16 "SECTORS/TRACK"
                        * HIGH NIBBLE IS "TRACK" EXTENDED ADDR A16-A20
                        * LO NIBBLE IS "SECTOR" A12-A15
EB34                    READV
EB34 34    73                   PSHS   CC,A,X,Y,U
EB36 1A    10                   SEI
EB38 10FF  EBAD                 STS    SAVSTK     SAVE STACK POINTER
EB3C 10CE  EBAF                 LDS    #TMPSTK    POINT TO NEW STACK TO AVOID PROBLEMS
EB40 8D    3B                   BSR    SETDAT
EB42 1E    12                   EXG    X,Y
EB44 8D    29                   BSR    COPYIT
EB46 20    10                   BRA    RESDAT
EB48                    WRITV
EB48 34    71                   PSHS   CC,X,Y,U
EB4A 1A    10                   SEI
EB4C 10FF  EBAD                 STS    SAVSTK
EB50 10CE  EBAF                 LDS    #TMPSTK
EB54 8D    27                   BSR    SETDAT
EB56 8D    17                   BSR    COPYIT
                        * RESTORE DAT
EB58                    RESDAT
EB58 B6    DFD0                 LDA    LRARAM     RESTORE SECTION NUMBER
EB5B B7    FFF0                 STA    DATRAM
EB5E B6    DFD2                 LDA    LRARAM+2   RESTORE SECTION 2
EB61 B7    FFF2                 STA    DATRAM+2
EB64 10FE  EBAD                 LDS    SAVSTK
EB68 35    71                   PULS   CC,X,Y,U
EB6A C6    00                   LDB    #0
EB6C 1C    FE                   CLC
EB6E 39                         RTS
                        * COPY BLOCK FROM X-PTR TO Y-PTR
                        * DESTROYS D,X,Y
```

```
EB6F                COPYIT
EB6F C6    80                   LDB     #128
EB71 34    04                   PSHS    B           COUNTER ON STACK
EB73                LOOP
EB73 EC    81                   LDD     ,X++
EB75 ED    A1                   STD     ,Y++
EB77 6A    E4                   DEC     ,S
EB79 26    F8                   BNE     LOOP
EB7B 35    84                   PULS    B,PC
                * SET THE DAT TO MAP REQUIRED SECTOR INTO PAGE 0 IF
                * THIS DOES NOT INTERFERE WITH FCB ELSE MAP TO
                * PAGE 2 AND POINT Y TO MAPPED SECTOR.
EB7D                SETDAT
EB7D 5A                         DECB                SECTOR NOS 0-15 (FROM 1-16)
EB7E C1    0F                   CMPB    #15         SEC/TRK
EB80 22    20                   BHI     SEEKER
EB82 B1    ED15                 CMPA    MAXTRK
EB85 22    1B                   BHI     SEEKER
EB87 108E  ED15                 LDY     #YDAT       ADD EXT
EB8B A6    A6                   LDA     A,Y
EB8D 8C    1000                 CMPX    #$1000      SEE IF INTERFERE WITH DATA MEM
EB90 25    05                   BLO     SD1
                * USE PAGE 0 SINCE FCB DATA NOT IN THIS AREA
EB92 B7    FFFC                 STA     DATRAM      SET DAT PAGE 0
EB95 20    05                   BRA     SD2
                * USE PAGE 2 SINCE FCB DATA IS IN PAGE 0
EB97                SD1
EB97 B7    FFFE                 STA     DATRAM+2
EB9A CB    20                   ADDB    #$20        SET FOR SECOND SECTION USE
EB9C                SD2
EB9C 1F    98                   TFR     B,A         COPY SECTOR (0-15) TO A
EB9E 5F                         CLRB
EB9F 1F    02                   TFR     D,Y         SET Y TO MEMORY ADDRESS OF SECTOR
EBA1 39                         RTS
                * ERROR IN SEEK (INVALID TRACK OR SECTOR)
EBA2                SEEKER
EBA2 10FE  EBAD                 LDS     SAVSTK
EBA6 35    71                   PULS    CC,X,Y,U    RESTORE IRQ
EBA8 C6    10                   LDB     #$10        SEEK ERROR
EBAA 1A    01                   SEC
EBAC 39                         RTS
EBAD                SAVSTK      RMB     2
EBAF                TEMPS       RMB     16
          EBBF      TMPSTK      EQU     *
EBBF                DUMRET
EBBF 5F                         CLRB
EBC0 1F    9B                   TFR     B,DP        CLEAR DP TO PATCH FLEX
EBC2                VERIFV
EBC2                RESTOV
EBC2                DRIVEV
EBC2                CKRDYV
EBC2                QUICKV
EBC2                DIRTYV

EBC2                DSEXIT
EBC2 5F                         CLRB                ABBY FOR NO ERRORS
EBC3 39                         RTS
                * THE VIRTUAL DISK DRIVER
                *
                *
                * HARD DISK DRIVERS

EBC4                VECVID
EBC4 7E    EC51                 JMP     HDREAD
EBC7 7E    EC89                 JMP     HWRITE
>EBCA 7E   EBE2                 JMP     HVERIF
>EBCD 7E   EBF7                 JMP     HRESTR
>EBD0 7E   EC12                 JMP     HDRIVE
>EBD3 7E   EBF1                 JMP     HCHECK
>EBD6 7E   EBF1                 JMP     HQCICK
>EBD9 7E   EBDC                 JMP     HINIT

                * THESE ARE ALL STANDARD FLEX EQUATES
          CD15      GETCHR      EQU     $CD15
          CD18      INBUFF      EQU     $CD18
          CD33      SETEXT      EQU     $CD33
          D406      FMS         EQU     $D406
          CD2D      GETFIL      EQU     $CD2D
          CD0F      OUTCH       EQU     $CD0F
          CD48      INDEC       EQU     $CD48
          CD42      GETHEX      EQU     $CD42
          CD45      OUTADR      EQU     $CD45
          CD4E      STAT        EQU     $CD4E
          CD3F      RPTERR      EQU     $CD3F
          CD3C      OUTHEX      EQU     $CD3C
          CD27      PSTCR       EQU     $CD27
          CC0E      DATE        EQU     $CC0E

          ED30      HBASE       EQU     $ED30       SYSTEM DEPENDENT: FIRST ADDRESS OF DISK PORT
          ED30      RDATA       EQU     HBASE       REGISTER 1 FOR READ PURPOSES
          ED30      WDATA       EQU     HBASE       REGISTER 1 FOR WRITE PURPOSES
          ED31      ERROR       EQU     HBASE+1     REGISTER 2, ETC. FOLLOW IN ORDER
          ED31      WPRE        EQU     HBASE+1     SEE WD1000 MANUAL FOR DEFINITION OF REGISTERS
          ED32      RSCNT       EQU     HBASE+2
          ED32      WSCNT       EQU     HBASE+2
          ED33      RSNUM       EQU     HBASE+3
          ED33      WSNUM       EQU     HBASE+3
          ED34      RCTLL       EQU     HBASE+4
          ED34      WCTLL       EQU     HBASE+4
          ED35      RCTLH       EQU     HBASE+5
          ED35      WCTLH       EQU     HBASE+5
          ED36      RSDH        EQU     HBASE+6
          ED36      WSDH        EQU     HBASE+6
          ED37      STATUS      EQU     HBASE+7
          ED37      COMAND      EQU     HBASE+7
          0007      BELL        EQU     7           ASCII CONTROL G FOR BELL SIGNAL
          ED04      PORT        EQU     $ED04       TERMINAL PORT ADDRESS
          0020      SECMAX      EQU     32
          00FF      TRKMAX      EQU     255
          0050      FORMAT      EQU     %01010000 uninverted for true data bus
          0020      FREDO       EQU     32
          0014      RESTSL      EQU     $14
          0010      REST        EQU     $10

                * INITIALIZE DRIVE AT WARMSTART

>EBDC 17   0017     HINIT       LBSR    RESTRE      cold start
EBDF 7E    CD03                 JMP     WARMS       AND RETURN TO FLEX WARMSTART ROUTINE

EBE2 C6    00       HVERIF      LDB     #0          DON'T CHECK FOR WRITE ERRORS
EBE4 1C    FE                   CLC
EBE6 39                         RTS
EBE7 86    07       HRESTR      LDA     #BELL       BEEP TERMINAL ON ERROR
EBE9 BD    CD18                 JSR     PUTCHR
EBEC C6    00                   LDB     #0          DRIVE DOES ITS OWN RECOVERY
EBEE 1C    FE                   ANDCC   #$FE
EBF0 39                         RTS
EBF1                HQCICK
EBF1 C6    00       HCHECK      LDB     #0          JUST A DUMMY DRIVE CHECK
EBF3 1C    FE                   CLC
EBF5 39                         RTS

                * INITIALIZE THE DRIVE

EBF6 86    00       RESTRE      LDA     #$00
EBF8 B7    ED36                 STA     WSDH        CHOOSE DRIVE,SIDE,HEAD AS ZERO
EBFB 86    20                   LDA     #FREDO
EBFD B7    ED31                 STA     WPRE        SET PRECOMPENSATION TRACK
EC00 86    00                   LDA     #$00
EC02 B7    ED35                 STA     WCTLH       CYLINDER HIGH IS ALWAYS ZERO ON 5 MEG DRIVE
EC05 86    10                   LDA     #REST
>EC07 17   0027                 LBSR    COMOUT      dummy restore to reset seek speed
EC0A 7D    ED37     REST2       TST     STATUS
EC0D 2B    FB                   BMI     REST2       BUSY?
EC0F C6    00                   LDB     #0
EC11 39                         RTS

                * DRIVE SELECT ROUTINE

EC12 A6    03       HDRIVE      LDA     3,X
EC14 81    01                   CMPA    #1
EC16 23    05                   BLS     TALDRV
EC18 C6    0F                   LDB     #$0F        ERROR RETURN
EC1A 1A    01                   SEC
EC1C 39                         RTS
EC1D 27    09       TALDRV      BEQ     ONE
EC1F 86    00                   LDA     #$00        DRIVE 0 IS TRANSLATED TO HEAD ZERO
EC21 B7    ED36                 STA     WSDH
EC24 5F                         CLRB
EC25 1C    FE                   CLC
EC27 39                         RTS
EC28 86    01       ONE         LDA     #$01        DRIVE 1 IS TRANSLATED AS HEAD ONE
EC2A B7    ED36                 STA     WSDH
EC2D 5F                         CLRB
EC2E 1C    FE                   CLC
EC30 39                         RTS

                * Send Command to WD1005

EC31 34    04       COMOUT      PSHS    B
EC33 F6    ED37     WAIT        LDB     STATUS
EC36 C5    80                   BITB    #$80        check for busy
EC38 26    F9                   BNE     WAIT        if busy, wait for it
EC3A C5    02                   BITB    #$02        check for command in progress
EC3C 26    03                   BNE     COMG        if so, error - return
EC3E B7    ED37                 STA     COMAND      issue command
EC41 35    84       COMG        PULS    B,PC
          00ED                  SETDP   HBASE/256 THIS IS NEEDED TO SUPPRESS WARNINGS AT ASSEMBLY

                * READ A SECTOR
                * A HAS TRACK, B HAS SECTOR, X HAS BUFFER ADDRESS

          0020      READ        EQU     $20
          0030      WRITE       EQU     $30

                * FIRST, A SUB COMMON TO HREAD AND HWRITE

EC43 34    02       PREDO       PSHS    A           SAVE TRACK TEMPORARILY
EC45 5A                         DECB                FIX SECTOR FOR FLEX
EC46 86    ED                   LDA     #HBASE/256 GET DIRECT PAGE OF DISK INTERFACE
EC48 1F    8B                   TFR     A,DP
EC4A 35    02                   PULS    A           RECOVER TRACK
EC4C D7    33                   STB     <WSNUM      GIVE IT THE SECTOR
EC4E 97    34                   STA     <WCTLL      AND THE TRACK
EC50 39                         RTS

                * NOW FOR THE READ PROPER

EC51 34    08       HDREAD      PSHS    DP          SAVE DIRECT PAGE
EC53 8D    EE                   BSR     PREDO       SET UP FOR A READ COMMAND
EC55 86    20                   LDA     #READ

EC57 8D    D8                   BSR     COMOUT
EC59 96    37       WAIT2       LDA     <STATUS
EC5B 85    80                   BITA    #$80        busy?
EC5D 26    FA                   BNE     WAIT2
EC5F D6    37       READLP      LDB     <STATUS     our read loop checks DRQ
EC61 C5    08                   BITB    #8          instead of counting to 256
EC63 27    06                   BEQ     RDONE
EC65 96    30                   LDA     <RDATA

EC67 A7    80                   STA     0,X+
EC69 20    F4                   BRA     READLP

EC6B 30    89 FF00  RDONE       LEAX    -256,X      RESET X FOR FLEX
EC6F D6    37                   LDB     <STATUS
EC71 C5    01                   BITB    #1          ERROR?
EC73 26    06                   BNE     READE
EC75 C6    00                   LDB     #0
EC77 1C    FE                   CLC
EC79 35    88                   PULS    DP,PC       RETURN WITH NO ERRORS FOUND
EC7B 96    31       READE       LDA     <ERROR      ERROR, CHECK IT
EC7D 81    40                   CMPA    #%01000000 DATA CRC ERROR?
EC7F 26    04                   BNE     READ6
EC81 C6    08                   LDB     #8          MUST BE CRC
EC83 35    88                   PULS    DP,PC
EC85 C6    10       READ6       LDB     #$10        ID ERROR FOR FLEX
EC87 35    88                   PULS    DP,PC

                * ROUTINE TO WRITE OUT A SECTOR TO DISK

EC89 34    08       HWRITE      PSHS    DP          SAVE DIRECT PAGE
>EC8B 17   FFB5                 LBSR    PREDO       THIS WORKS JUST ABOUT LIKE "HREAD"
EC8E 86    30                   LDA     #WRITE
>EC90 17   FF9E                 LBSR    COMOUT
EC93 A6    80       PUTLP       LDA     0,X+
EC95 D6    37       PUTLP1      LDB     <STATUS
EC97 C5    08                   BITB    #$08        data request?
EC99 26    1C                   BNE     WRDTY       yes
EC9B C5    80                   BITB    #$80        no, test busy
EC9D 26    F6                   BNE     PUTLP1

EC9F 30    89 FF00  WDONE       LEAX    -256,X
ECA3 0D    37       WRITE       TST     <STATUS
ECA5 2B    FC                   BMI     WRITE
```

```
ECA7 B6    37             LDA    <STATUS
ECA9 C5    01             BITB   #1         READY?
ECAB 27    09             BEQ    WAIT?
ECAD C4    10             LDB    #%00010000 MUST BE DATA FIELD
ECAF 20    04             BRA    WEXTB
ECB1 C6    00     WAIT?   LDB    #0
ECB3 1C    FE             CLC
ECB5 35    04     WEXTB   PULS   DP,PC

ECB7 97    30     WEXIT   STA    CMDATA
ECB9 20    08             BRA    FOTLP

      0000                SETDP  0          restore direct page

                  * End of Hardware drivers
                  *

                  * HDC HARD DISK DRIVERS
                  *
                  *
                  * GIMIX CALLS
              E856 GIMDSK  SET    $E856
ECBB              GVECTB
ECBB 34    06             PSHS   D
ECBD EC    62             LDD    2,S        get return address
ECBF C3    F137           ADDD   #$DE00-VECDIR convert to $DE00 org
ECC2 ED    62             STD    2,S        put back on stack
ECC4 35    06             PULS   D          restore
ECC6 7E    E856           JMP    GIMDSK
                  *
                  * vector table for GIMIX
                  *
ECC9              VECDIR
>ECC9 BD   ECBB           JSR    GVECTB
>ECCC BD   ECBB           JSR    GVECTB
>ECCF BD   ECBB           JSR    GVECTB
>ECD2 BD   ECBB           JSR    GVECTB
>ECD5 BD   ECBB           JSR    GVECTB
>ECD8 BD   ECBB           JSR    GVECTB
>ECDB BD   ECBB           JSR    GVECTB
ECDE              DCOLD
>ECDE BD   ECBB           JSR    GVECTB
ECE1              DWARM
>ECE1 BD   ECBB           JSR    GVECTB
>ECE4 BD   ECBB           JSR    GVECTB
ECE7              COLD
ECE7 8D    F5             BSR    DCOLD
ECE9 BD    EB23           JSR    VECTIB+$15 INCLUDE NEW DRIVER
ECEC BD    EA09           JSR    VECDIR+$15 init vdiskmaster
ECEF 39                   RTS
ECF0 8D    EF     WARM    BSR    DWARM
ECF2 BD    EB2E           JSR    VECTIB+$18 INCLUDE NEW DRIVER
ECF5 39                   RTS
                  *
                  *
                  * Put SET.CMD into user command table
                  *
ECF6 53 45 54 00  CMDTBL  FCC    'SET',0
ECFA E824                 FDB    SET
ECFC 00                   FCC    0
                  *
ECFD ECFD         ZZZZZZ  FDB    *          FLAG LAST ASSEMBLED ADDRESS
                  *
                  *
CC12                      ORG    USRCMD
CC12 ECF6                 FDB    CMDTBL
                  *

                  *

                  *
                  * LAST TO LOAD IS THE VECTOR TABLE
C100                      ORG    $C100      FLEX .CMD AREA
                  * INSTALL DISK DRIVER VECTOR TABLE
C100              START
                  *  Save initial work and system drive numbers
C100 B6    CC0B           LDA    SYSDRV
C103 B7    E813           STA    SYSSAV
C106 B6    CC0C           LDA    WRDRV
C109 B7    E814           STA    WRSAV
                  * Find all unused memory for virtual disk use.
                  * First clear the FDAT table
C10C 8E    E815           LDX    #FDAT
C10F 5F                   CLRB              256 bytes long
C110 6F    80     CLRLP   CLR    ,X+
C112 5A                   DECB
C113 26    FB             BNE    CLRLP
                  * Now tag pages used by FLEX with number 2
C115 108E DFD0            LDY    #LRARAM
C119 86    10             LDA    #16        Maximum of 16 pages for FLEX
C11B E6    A0     FLEXLP  LDB    ,Y+
C11D 27    08             BEQ    LOOK       abort table
C11F 8E    E815           LDX    #FDAT
C122 3A                   ABX
C123 6C    84             INC    ,X
C125 6C    84             INC    ,X
C127 4A                   DECA
C128 26    F1             BNE    FLEXLP
                  * Now rummage around looking for excess memory
C12A 5F           LOOK    CLRB
C12B 8E    E815   LOLP    LDX    #FDAT
C12E 3A                   ABX
C12F 6D    84             TST    ,X         flex memory?
C131 26    16             BNE    NXTLK      already tagged as FLEX memory
C133 F7    FFF0           STB    DATRAM     move page into address space 0-4k
C136 96    00             LDA    $0
C138 B7    E816           STA    XTEMP      save byte for return
C13B 43                   COMA
C13C 97    00             STA    $0
C13E 91    00             CMPA   $0         any memory there?
C140 26    07             BNE    NXTLK      no, go on with test
C142 6C    84             INC    ,X         tag with number 1
C144 B6    E816           LDA    XTEMP      replace byte
C147 97    00             STA    $0
C149 5C           NXTLK   INCB
C14A 26    DF             BNE    LOLP
C14C B6    DFD0           LDA    LRARAM     replace flex page
C14F B7    FFF0           STA    DATRAM
                  * Now squeeze table together and count pages for MAXTRK
C15  108E E815            LDY    #FDAT
C159 8E    E815           LDX    #FDAT
C15? 5F                   CLRB
C15A 7F    E815           CLR    MAXTRK
C15D A6    80     MEMLP   LDA    ,X+
C15F 81    01             CMPA   #1         Usable memory?
C161 26    05             BNE    NXTMEM     No, go on
C163 E7    A0             STB    ,Y+
C165 7C    E815           INC    MAXTRK
C168 5C           NXTMEM  INCB
C169 26    F2             BNE    MEMLP
                  * Now report to operator
C16B 7D    E815           TST    MAXTRK     see if any memory was found
C16E 26    0C             BNE    MEMFND
C170 8E    C1EF           LDX    #NOMEM
C173 BD    CD1E           JSR    PSTRNG
C176 BD    CD24           JSR    PCRLF
C179 7E    CD03           JMP    WARMS
C17C BD    CD24   MEMFND  JSR    PCRLF
C17F 8E    C212           LDX    #REPORT
C182 BD    CD1E           JSR    PSTRNG
C185 B6    E815           LDA    MAXTRK
C188 C6    10             LDB    #16
C18A 3D                   MUL               number of usable sectors
C18B FD    E816           STD    XTEMP
C18E 7A    E815           DEC    MAXTRK     maximum track number
C191 5F                   CLRB
C192 8E    E816           LDX    #XTEMP
C195 BD    CD39           JSR    OUTDEC
C198 BD    CD24           JSR    PCRLF
C19B B6    E815           LDA    MAXTRK     Move max track to TMAX
C19E B7    EB04           STA    TMAX
                  * CHECK FOR JMP INSTRUCTION IN DISK DRIVE TABLE
                  * IF NOT JMP THEN MUST BE GIMIX JSR CALL
                  * WHICH SHOULD BE CORRECTED IN THIS SOURCE
C1A1 86    7E             LDA    #$7E       "JMP"
C1A3 C6    09             LDB    #9         # ENTRIES IN TABLE
C1A5 8E    DE00           LDX    #DSVTBL
C1A8 A1    84     SU2     CMPA   ,X
C1AA 26    15             BNE    CPYDSV     SKIP IF JSR
C1AC 30    03             LEAX   3,X
C1AE 5A                   DECB
C1AF 26    F7             BNE    SU2
                  * COPY OLD TBC/DSVPC TABLE TO VECDIR
C1B1 8E    DE00           LDX    #DSVTBL
C1B4 108E ECC9            LDY    #VECDIR
C1B8 C6    1E             LDB    #(ENDTBL-JMPTBL)
C1BA A6    80     SU4     LDA    ,X+
C1BC A7    A0             STA    ,Y+
C1BE 5A                   DECB
C1BF 26    F9             BNE    SU4
                  * COPY NEW TABLE TO $DE00
C1C1              CPYDSV
C1C1 8E    C1D4           LDX    #JMPTBL
C1C4 108E DE00            LDY    #DSVTBL
C1C8 C6    1E             LDB    #(ENDTBL-JMPTBL)
C1CA A6    80     SU6     LDA    ,X+
C1CC A7    A0             STA    ,Y+
C1CE 5A                   DECB
C1CF 26    F9             BNE    SU6
C1D1 7E    E824           JMP    SET
C1D4              JMPTBL
C1D4 BD    EA88           JSR    DISKRD     READ SECTOR
C1D7 BD    EA88           JSR    DISKWD     WRITE SECTOR
C1DA BD    EA8E           JSR    DISKVD     VERIFY SECTOR
C1DD BD    EAA5           JSR    DRIVE      RESTOR HEAD
C1E0 BD    EAA5           JSR    DRIVE      SELECT DRIVE
C1E3 BD    EAA5           JSR    DRIVE      CHECK DRIVE READY
C1E6 BD    EAA5           JSR    DRIVE      QUICK CHECK DRIVE READY
                  * NOTE NEXT ARE JMP INSTEAD OF JSR
C1E9 7E    ECE7           JMP    COLD       COLD START
C1EC 7E    ECF0           JMP    WARMD      WARMSTART
                  * THIS LAST ENTRY IS NOT USED BY FLEX AND MAY NOT BE PRESENT
                  * IN YOUR VERSION OF FLEX
                  * INCLUDE ONLY IF REFERENCED BY ADDED
                  * DRIVER(S) TO THE FLEX TABLE AT $DE1E
                  *
                  * JSR DISKIO SEEK TRACK
              C1EF ENDTBL  EQU    *
                  *
C1EF 4E 6F 20 6D  NOMEM   FCC    'No memory found - Process aborted ',$4
C1F3 65 6D 6F 72
C1F7 79 20 66 6F
C1FB 75 6E 64 20
C1FF 2D 20 50 72
C203 6F 63 65 73
C207 73 20 61 62
C20B 6F 72 74 65
C20F 64 20 04
C212 4D 61 78 69  REPORT  FCC    'Maximum number of virtual sectors found = ',$4
C216 6D 75 6D 20
C21A 6E 75 6D 62
C21E 65 72 20 6F
C222 66 20 76 69
C226 72 74 75 61
C22A 6C 20 73 65
C22E 63 74 6F 72
C232 73 20 66 6F
C236 75 6E 64 20
C23A 3D 20 04
                          END    START

0 ERROR(S) DETECTED

SYMBOL TABLE:

BELL  0007  CHECK  E837  CK1   E0EF  CK2   E0FD  CK4   EA06
CK6   EA1E  CK8    EA22  CKRDTY EBC2 CKSTAT CDAF  CLRLP C110
CMDTBL ECF6 COLD   ECE7  COMS  ECE1  COMAND E037  COMOUT EC 1
COPYXX EB6F CPYDSV C1C1  DATE  CC0E  DATRAM FFF0  DISIT? EBC2
DIO1  EAB1  DIO2   EABD  DIO4  EB0B  DIOVEC EB0B  DISKIO EA8E
DRIVE EAA5  DRIVET EB0A  DRIVEV EB0A DSKRTN EAFD  DSVTBL DE00
DSVTOT 0003 DSKEXV EBC2  DWARMV EB8F ENDTBL C1EF  ERRORS EB0C
ERRORS E031 FINI   EB0C  FLEX  E0D8  FLEXLP C11B  FMS   D400
FORMAT 0050 GCOLD  ECDE  GDRIVE EB07 GETCHR CD15  GETFIL CD2D
GETXXX CD42 GIMDSK E856  GVECTB ECBB GWARM ECE1   GWASE E830
HCHECK EAF1 HDRIVE EC12  HINIT EBAC  HQUICK EAF1  HREAD EC51
HRESTR EB87 HVERIF EB87  HWRITE EC09 INBUFF CD1B  INCHD CD0D
INDEC CD48  JMPTBL C1D4  JMPVEN EB04 LDRIVE EB08  LDRIVE EB08
LOLP  C12B  LOOK   C12A  LOOP  E873  LRARAM DFD0  LSTERR CC11
MAXTRK E815 MEMFND C17C  MEMLP C15D  MEMORY EB00  MSG   EA99
NDRVES EB06 NEXTCT EB09  NEXT  EBAF  NOMEM C1EF   NXTCH CD27
NXTCH CD27  NXTLK  C149  NXTMEM C168 NXTR  EBD5   ONE   EC28
OUTADR CD45 OUTCH  CD0F  OUTCH2 CD13 OUTDEC CD39  OUTHEX CD3C
```



To Be Continued Next Month

"CRYPTOQUOTE"

By: Mickey B. Ferguson

Continued From Last Month

```
156   1094 3D                    MUL
159   1095 C3   005E             ADDD   #B_STR-MAXLEN
160   1098 BD   14D9             JSR    GETSTR
161   109B CE   076F             LDU    #ST_BUF
162   109E C6   50               LDB    #MAXLEN
163   10A0 BD   152B             JSR    MOVVR1
164   10A3 8E   156B             LDX    #SPACE
165   10A6 BD   1529             JSR    MOVVAR
166   10A9 CE   076F             LDU    #ST_BUF
167   10AC 35   10               PULS   X
168   10AE BD   13D7             JSR    STNG_2
169
170                        * 0110 NEXT Y
171
172   10B1 BD   13C6             JSR    NEXTY
173   10B4 2D   D0               BLT    LN_100
174   10B6 26   03               BNE    LN_120
175   10B8 5D                    TSTB
176   10B9 27   CB               BEQ    LN_100
177
178                        * 0120 GOTO 40
179
180   10BB 20   8A         LN_120  BRA    LN__40
181
182                        * 0130 LET Z=X-1
183
184   10BD 9E   08         LN_130  LDX    X_VAR
185   10BF 30   1F                 LEAX   -1,X
186   10C1 9F   0C                 STX    Z_VAR
187
188                        * 0140 PRINT CHR$(12);
189
190   10C3 86   0C         LN_140  LDA    #12
191   10C5 BD   13EC               JSR    OUTEEE
192
193                        * 0150 FOR X=1 TO Z
194
195   10C8 BD   13B2       LN_150  JSR    X_12_Z
196
197                        * 0160 PRINT B$(X)
198
199   10CB BD   1442       LN_160  JSR    SETBUF
200   10CE D6   09                 LDB    X_VAR+1
201   10D0 86   50                 LDA    #MAXLEN
202   10D2 3D                      MUL
203   10D3 C3   005E               ADDD   #B_STR-MAXLEN
204   10D6 BD   1409               JSR    GETSTR
205   10D9 C6   50                 LDB    #MAXLEN
206   10DB BD   1479               JSR    MOVST1
207   10DE BD   1453               JSR    PST_CR
208
209                        * 0170 PRINT C$(X)
210
211   10E1 BD   1442               JSR    SETBUF
212   10E4 D6   09                 LDB    X_VAR+1
213   10E6 86   50                 LDA    #MAXLEN
214   10E8 3D                      MUL
215   10E9 C3   037E               ADDD   #C_STR-MAXLEN
216   10EC BD   14D9               JSR    GETSTR
217   10EF C6   50                 LDB    #MAXLEN
218   10F1 BD   1479               JSR    MOVST1
219   10F4 BD   1453               JSR    PST_CR
220
221                        * 0180 PRINT
222
223   10F7 BD   1455               JSR    PCRLF
224
225                        * 0190 NEXT X
226
227   10FA BD   13BC               JSR    NEXTX
228   10FD 2D   CC                 BLT    LN_160
229   10FF 26   03                 BNE    LN_200
230   1101 5D                      TSTB
231   1102 27   C7                 BEQ    LN_160
232
233                        * 0200 PRINT
234
235   1104 BD   1455       LN_200  JSR    PCRLF
236
237                        * 0210 INPUT BUF$
238
239   1107 BD   13EF               JSR    INPUT
240
241                        * 0220 IF BUF$="STOP" THEN 600
242
243   110A 8E   06EE               LDX    #IO_BUF
244   110D CE   076F               LDU    #ST_BUF
245   1110 BD   1529               JSR    MOVVAR
246   1113 BD   1546               JSR    EQUALS
247   1116 DF   04                 STU    X_TEMP
248   1118 8E   156D               LDX    #STOP
249   111B BD   1529               JSR    MOVVAR
250   111E 8E   076F               LDX    #ST_BUF
251   1121 BD   154A               JSR    IF
252   1124 1025 0240               LBCS   LN_600
253
254                        * 0230 IF BUF$="RESET" THEN 440
255
256   1128 8E   06EE               LDX    #IO_BUF
257   112B CE   076F               LDU    #ST_BUF
258   112E BD   1529               JSR    MOVVAR
259   1131 BD   1546               JSR    EQUALS
260   1134 DF   04                 STU    X_TEMP
261   1136 8E   1572               LDX    #RESET
262   1139 BD   1529               JSR    MOVVAR
263   113C 8E   076F               LDX    #ST_BUF
264   113F BD   154A               JSR    IF
265   1142 1025 013F               LBCS   LN_440
266
267                        *      LET D$=BUF$
268
269   1146 CE   06EE               LDU    #IO_BUF
270   1149 8E   000E               LDX    #D_STR
271   114C BD   13D7               JSR    STNG_2
272
273                        * 0260 IF MID$(D$,2,1)<>"*" THEN 380
274
275   114F CE   076F               LDU    #ST_BUF
276   1152 32   7E                 LEAS   -2,S
277   1154 BD   14DE               JSR    NOT_EQ
278   1157 8E   000E               LDX    #D_STR
279   115A C6   50                 LDB    #MAXLEN
280   115C BD   152B               JSR    MOVVR1
281   115F C6   02                 LDB    #2
282   1161 34   04                 PSHS   B
283   1163 C6   01                 LDB    #1
284   1165 35   02                 PULS   A
285   1167 BD   14F6               JSR    MID
286   116A 8E   076F               LDX    #ST_BUF
287   116D BD   1546               JSR    EQUALS
288   1170 DF   04                 STU    X_TEMP
289   1172 8E   1518               LDX    #EQUAL
290   1175 BD   1529               JSR    MOVVAR
291   1178 8E   076F               LDX    #ST_BUF
292   117B BD   154A               JSR    IF
293   117E 1024 00D6               LBCC   LN_380
294
295                        * 0270 FOR X=1 TO Z
296
297   1182 BD   13B2               JSR    X_12_Z
```

```
298
299                             * 0280 LET E$=""
300
301   1185 8E   005E      LN_280  LDX    #E_STR
302   1188 BD   13DC              JSR    STNG20
303
304                             * 0290 FOR Y=1 TO LEN(C$(X))
305
306   118B BD   1391              JSR    Y_CDX
307
308                             * 0300 IF MID$(C$(X),Y,1)<>LEFT$(D$,1) THEN 330
309
310   118E CE   076F      LN_300  LDU    #ST_BUF
311   1191 32   7E               LEAS   -2,S
312   1193 BD   14DE              JSR    MOT_EQ
313   1196 D6   09                LDB    X_VAR+1
314   1198 86   50                LDA    #MAXLEN
315   119A 3D                     MUL
316   119B C3   037E              ADDD   #C_STR-MAXLEN
317   119E BD   14D9              JSR    GETSTR
318   11A1 C6   50                LDB    #MAXLEN
319   11A3 BD   152B              JSR    MOVVR1
320   11A6 D6   0B                LDB    Y_VAR+1
321   11A8 34   04                PSHS   B
322   11AA C6   01                LDB    #1
323   11AC 35   02                PULS   A
324   11AE BD   14F6              JSR    MID
325   11B1 8E   076F              LDX    #ST_BUF
326   11B4 BD   1546              JSR    EQUALS
327   11B7 DF   04                STU    X_TEMP
328   11B9 32   7E                LEAS   -2,S
329   11BB BD   14DE              JSR    MOT_EQ
330   11BE 8E   000E              LDX    #D_STR
331   11C1 C6   50                LDB    #MAXLEN
332   11C3 BD   152B              JSR    MOVVR1
333   11C6 CC   0001              LDD    #1
334   11C9 BD   150F              JSR    LEFT
335   11CC 8E   076F              LDX    #ST_BUF
336   11CF BD   154A              JSR    1F
337   11D2 24   2E                BCC    LN_330
338
339                             * 0310 LET E$=E$+MID$(D$,3,11
340
341   11D4 8E   005E              LDX    #E_STR
342   11D7 CE   076F              LDU    #ST_BUF
343   11DA C6   50                LDB    #MAXLEN
344   11DC BD   152B              JSR    MOVVR1
345   11DF 32   7E                LEAS   -2,S
346   11E1 BD   14DE              JSR    MOT_EQ
347   11E4 8E   000E              LDX    #D_STR
348   11E7 C6   50                LDB    #MAXLEN
349   11E9 BD   152B              JSR    MOVVR1
350   11EC C6   03                LDB    #3
351   11EE 34   04                PSHS   B
352   11F0 C6   01                LDB    #1
353   11F2 35   02                PULS   A
354   11F4 BD   14F6              JSR    MID
355   11F7 CE   076F              LDU    #ST_BUF
356   11FA 8E   005E              LDX    #E_STR
357   11FD BD   13D7              JSR    STNC_2
358
359                             * 0320 GOTO 340
360
361   1200 20   34                BRA    LN_340
362
363                             * 0330 LET E$=E$+MID$(B$(X),Y,1)
364
365   1202 8E   005E      LN_330  LDX    #E_STR
366   1205 CE   076F              LDU    #ST_BUF
367   1208 C6   50                LDB    #MAXLEN
368   120A BD   152B              JSR    MOVVR1
369   120D 32   7E                LEAS   -2,S
370   120F BD   14DE              JSR    MOT_EQ
371   1212 D6   09                LDB    X_VAR+1
372   1214 86   50                LDA    #MAXLEN
373   1216 3D                     MUL
374   1217 C3   005E              ADDD   #B_STR-MAXLEN
375   121A BD   14D9              JSR    GETSTR
376   121D C6   50                LDB    #MAXLEN
377   121F BD   152B              JSR    MOVVR1
378   1222 D6   0B                LDB    Y_VAR+1
379   1224 34   04                PSHS   B
380   1226 C6   01                LDB    #1
381   1228 35   02                PULS   A
382   122A BD   14F6              JSR    MID
383   122D CE   076F              LDU    #ST_BUF
384   1230 8E   005E              LDX    #E_STR
385   1233 BD   13D7              JSR    STNC_2
386
387                             * 0340 NEXT Y
388
389   1236 BD   13C6      LN_340  JSR    NEXTY
390   1239 102D FF51              LBLT   LN_300
391   123D 26   05                BNE    LN_350
392   123F 5D                     TSTB
393   1240 1027 FF4A              LBEQ   LN_300
394
395                             * 0350 LET B$(X)=E$
396
397   1244 BD   136B      LN_350  JSR    LBDX
398
399                             * 0360 NEXT X
400
401   1247 BD   13BC              JSR    NEXTX
402   124A 102D FF37              LBLT   LN_280
403   124E 26   05                BNE    LN_370
404   1250 5D                     TSTB
405   1251 1027 FF30              LBEQ   LN_280
406
407                             * 0370 GOTO 140
408
409   1255 7E   10C3      LN_370  JMP    LN_140
410
411                             * 0380 PRINT CHR$(12);"WHAT";CHR$(7)
412
413   1258 BD   1442      LN_380  JSR    SETBUF
414   125B CE   076F              LDU    #ST_BUF
415   125E CC   000C              LDD    #12
416   1261 BD   1541              JSR    CHR
417   1264 8E   076F              LDX    #ST_BUF
418   1267 BD   1477              JSR    MOVSTR
419   126A 8E   157A              LDX    #WHAT
420   126D BD   1477              JSR    MOVSTR
421   1270 CE   076F              LDU    #ST_BUF
422   1273 CC   0007              LDD    #7
423   1276 BD   1541              JSR    CHR
424   1279 8E   076F              LDX    #ST_BUF
425   127C BD   1477              JSR    MOVSTR
426   127F BD   1453              JSR    PST_CR
427
428                             * 0390 GOTO 150
429
430   1282 7E   10C8              JMP    LN_150
431
432                             * 0440 FOR X=1 TO 2
433
434   1285 BD   13B2      LN_440  JSR    X_12_2
435
436                             * 0450 LET E$=""
437
438   1288 8E   005E      LN_450  LDX    #E_STR
439   128B BD   13DC              JSR    STNC20
440
441                             * 0460 FOR Y=1 TO LEN(C$(X))
442
443   128E BD   1391              JSR    Y_CDX
444
445                             * 0470 LET E$=E$+" "
446
447   1291 8E   005E      LN_470  LDX    #E_STR
448   1294 CE   076F              LDU    #ST_BUF
449   1297 C6   50                LDB    #MAXLEN
450   1299 BD   152B              JSR    MOVVR1
451   129C 8E   156B              LDX    #SPACE
452   129F BD   1529              JSR    MOVVAR
453   12A2 CE   076F              LDU    #ST_BUF
454   12A5 8E   005E              LDX    #E_STR
455   12A8 BD   13D7              JSR    STNG_2
456
457                             * 0480 NEXT Y
458
459   12AB BD   13C6              JSR    NEXTY
460   12AE 2D   E1                BLT    LN_470
461   12B0 26   03                BNE    LN_490
462   12B2 5D                     TSTB
463   12B3 27   DC                BEQ    LN_470
464
465                             * 0490 LET B$(X)=E$
466
467   12B5 BD   136B      LN_490  JSR    LBDX
468
469                             * 0500 NEXT X
```

```
12B8 BD   13BC            JSR    NEXTX
12BB 2D   CB              BLT    LN_450
12BD 26   03              BNE    LN_510
12BF 5D                   TSTB
12C0 27   C6              BEQ    LN_450

                 * 0510 GOTO 140

12C2 7E   10C3   LN_510   JMP    LN_140

                 * 0520 LET D$=""

12C5 8E   000E   LN_520   LDX    #D_STR
12C8 BD   13DC            JSR    STNG20

                 * 0530 FOR Y=1 TO LEN(C$(X))

12CB BD   1391            JSR    Y_CDX

                 * 0540 LET D$=D$+MID$(C$(X),Y,1)

12CE 8E   000E   LN_540   LDX    #D_STR
12D1 CE   076F            LDU    #ST_BUF
12D4 C6   50              LDB    #MAXLEN
12D6 BD   152B            JSR    MOVVR1
12D9 32   7E              LEAS   -2,S
12DB BD   14DE            JSR    NOT_EQ
12DE D6   09              LDB    X_VAR+1
12E0 86   50              LDA    #MAXLEN
12E2 3D                   MUL
12E3 C3   037E            ADDD   #C_STR-MAXLEN
12E6 BD   14D9            JSR    GETSTR
12E9 C6   50              LDB    #MAXLEN
12EB BD   152B            JSR    MOVVR1
12EE D6   0B              LDB    Y_VAR+1
12F0 34   04              PSHS   B
12F2 C6   01              LDB    #1
12F4 35   02              PULS   A
12F6 BD   14F6            JSR    MID
12F9 CE   076F            LDU    #ST_BUF
12FC 8E   000E            LDX    #D_STR
12FF BD   13D7            JSR    STNG_2

                 * 0550 LET D$=D$+" "

1302 8D   74              BSR    DD_SPC

                 * 0560 IF MID$(D$,LEN(D$)-1,1)<>" " THEN 570

1304 CE   076F            LDU    #ST_BUF
1307 32   7E              LEAS   -2,S
1309 BD   14DE            JSR    NOT_EQ
130C 8E   000E            LDX    #D_STR
130F C6   50              LDB    #MAXLEN
1311 BD   152B            JSR    MOVVR1
1314 32   7E              LEAS   -2,S
1316 BD   14DE            JSR    NOT_EQ
1319 8E   000E            LDX    #D_STR
131C C6   50              LDB    #MAXLEN
131E BD   152B            JSR    MOVVR1
1321 AE   E4              LDX    0,S
1323 BD   14E1            JSR    LEN
1326 BD   151D            JSR    RIGHT
1329 83   0001            SUBD   #$0001
132C 34   04              PSHS   B
132E C6   01              LDB    #1
1330 35   02              PULS   A
1332 BD   14F6            JSR    MID
1335 8E   076F            LDX    #ST_BUF
1338 BD   1546            JSR    EQUALS
133B DF   04              STU    X_TEMP
133D 8E   1568            LDX    #SPACE
1340 BD   1529            JSR    MOVVAR
1343 8E   076F            LDX    #ST_BUF
1346 BD   154A            JSR    IF
1349 24   02              BCC    LN_570

                 * 0565 LET D$=D$+" "

134B 8D   2B              BSR    DD_SPC

                 * 0570 NEXT Y

134D 8D   77     LN_570   BSR    NEXTY
134F 102D FF7B            LBLT   LN_540
1353 26   05              BNE    LN_580
1355 5D                   TSTB
1356 1027 FF74            LBEQ   LN_540

                 * 0580 LET C$(X)=D$

135A D6   09     LN_580   LDB    X_VAR+1
135C 86   50              LDA    #MAXLEN
135E 3D                   MUL
135F C3   037E            ADDD   #C_STR-MAXLEN
1362 8E   000E            LDX    #D_STR
1365 8D   6C              BSR    STNG_1

                 * 0590 RETURN

1367 39                   RTS

                 * 0600 STOP

1368 7E   CD03   LN_600   JMP    WARMS

                 * Start of run-time package

136B D6   09     LBDX     LDB    X_VAR+1
136D 86   50              LDA    #MAXLEN
136F 3D                   MUL
1370 C3   005E            ADDD   #B_STR-MAXLEN
1373 8E   005E            LDX    #E_STR
1376 20   5B              BRA    STNG_1

1378 8E   000E   DD_SPC   LDX    #D_STR
137B CE   076F            LDU    #ST_BUF
137E C6   50              LDB    #MAXLEN
1380 BD   1528            JSR    MOVVR1
1383 8E   156B            LDX    #SPACE
1386 BD   1529            JSR    MOVVAR
1389 CE   076F            LDU    #ST_BUF
138C 8E   000E            LDX    #D_STR
138F 20   46              BRA    STNG_2

1391 CC   0001   Y_CDX    LDD    #1
1394 DD   0A              STD    Y_VAR
1396 D6   09              LDB    X_VAR+1
1398 86   50              LDA    #MAXLEN
139A 3D                   MUL
139B C3   037E            ADDD   #C_STR-MAXLEN
139E BD   14D9            JSR    GETSTR
13A1 CE   076F            LDU    #ST_BUF
13A4 C6   50              LDB    #MAXLEN
13A6 BD   1528            JSR    MOVVR1
13A9 8E   076F            LDX    #ST_BUF
13AC BD   14E1            JSR    LEN
13AF DD   02              STD    NEXT_Y
13B1 39                   RTS

13B2 CC   0001   X_12_Z   LDD    #1
13B5 DD   08              STD    X_VAR
13B7 DC   0C              LDD    Z_VAR
13B9 DD   00              STD    NEXT_X
13BB 39                   RTS

13BC DC   08     NEXTX    LDD    X_VAR
13BE C3   0001            ADDD   #$0001
13C1 DD   08              STD    X_VAR
13C3 93   00              SUBD   NEXT_X
13C5 39                   RTS

13C6 DC   0A     NEXTY    LDD    Y_VAR
13C8 C3   0001            ADDD   #$0001
13CB DD   0A              STD    Y_VAR
13CD 93   02              SUBD   NEXT_Y
13CF 39                   RTS

13D0 8E   156C   STNG_0   LDX    #ZERO
13D3 1F   13     STNG_1   TFR    X,U
13D5 1F   01              TFR    D,X
13D7 C6   50     STNG_2   LDB    #MAXLEN
13D9 7E   14EC            JMP    LET

13DC CE   156C   STNG20   LDU    #ZERO
13DF 20   F6              BRA    STNG_2

13E1 A6   80     PDATA1   LDA    0,X+
13E3 81   04              CMPA   #4
13E5 26   01              BNE    PDATA
13E7 39                   RTS
```

```
642   13E0 8D   02      PDATA   BSR   OUTEEE
643   13E2 20   F5              BRA   PDATA1
644
645   13E4 7E   CD1B    OUTEEE  JMP   PUTCHR
646
647   13E7 86   3F      INPUT   LDA   #$3F
648   13E9 8D   F9              BSR   OUTEEE
649   13EB 86   20              LDA   #$20
650   13ED 8D   F5              BSR   OUTEEE
651   13EF 8D   38      INPUT1  BSR   INPUTS
652   13F1 C6   80              LDB   #$80
653   13F3 8D   CD15    INPUT2  JSR   GETCHR
654   13F6 B1   CC00            CMPA  BS_CHR
655   1401 26   09              BNE   INPUT3
656   1403 C1   80              CMPB  #$80
657   1405 27   0A              BEQ   INPUT4
658   1407 5C                   INCB
659   1408 30   1F              LEAX  -1,X
660   140A 20   EF              BRA   INPUT2
661   140C B1   CC01    INPUT3  CMPA  DL_CHR
662   140F 26   09              BNE   INPUT5
663   1411 8E   157F    INPUT4  LDX   #DELSTR
664   1414 8D   CB              BSR   PDATA1
665   1416 8D   0C              BSR   INPUT6
666   1418 20   DD              BRA   INPUT1
667   141A A7   84      INPUT5  STA   0,X
668   141C 81   0D              CMPA  #$0D
669   141E 26   09              BNE   INPUT7
670   1420 6F   80              CLR   0,X+
671   1422 9F   06              STX   BUFPNT
672   1424 8E   1586    INPUT6  LDX   #CRLF
673   1427 20   B8              BRA   PDATA1
674   1429 5D           INPUT7  TSTB
675   142A 27   CF              BEQ   INPUT2
676   142C 30   01              LEAX  1,X
677   142E 5A                   DECB
678   142F 20   CA              BRA   INPUT2
679   1431 8E   06EE    INPUTS  LDX   #IO_BUF
680   1434 39                   RTS
681
682   1435 86   20      MOVSPC  LDA   #$20
683   1437 109E 06      MOVCHR  LDY   BUFPNT
684   143A A7   A0              STA   0,Y+
685   143C 6F   A4              CLR   0,Y
686   143E 109F 06              STY   BUFPNT
687   1441 39                   RTS
688
689   1442 8E   06EE    SETBUF  LDX   #IO_BUF
690   1445 9F   06              STX   BUFPNT
691   1447 39                   RTS
692
693   1448 8D   E7      PSTRNG  BSR   INPUTS
694   144A A6   80      PSTNG1  LDA   0,X+
695   144C 27   04              BEQ   EXIT
696   144E 8D   9C              BSR   OUTEEE
697   1450 20   F8              BRA   PSTNG1
698   1452 39           EXIT    RTS
699
700   1453 8D   F3      PST_CR  BSR   PSTRNG
701   1455 8E   1586    PCRLF   LDX   #CRLF
702   1458 A6   80      PCRLF1  LDA   0,X+
703   145A 81   04              CMPA  #4
704   145C 27   F4              BEQ   EXIT
705   145E 8D   BC              BSR   OUTEEE
706   1460 20   F6              BRA   PCRLF1
707
708   1462 34   02      BIAS    PSHS  A
709   1464 A6   84              LDA   0,X
710   1466 26   04              BNE   BIAS1
711   1468 6D   01              TST   1,X
712   146A 27   06              BEQ   BIAS2
713   146C BA   30      BIAS1   ORA   #$30
714   146E 8D   C7              BSR   MOVCHR
715   1470 6C   01              INC   1,X
716   1472 6F   84      BIAS2   CLR   0,X
717   1474 35   02              PULS  A
718   1476 39                   RTS
719
720   1477 C6   80      MOVSTR  LDB   #$80
721   1479 A6   80      MOVST1  LDA   0,X+
722   147B 27   05              BEQ   MVEXIT
723   147D 8D   B8              BSR   MOVCHR
724   147F 5A                   DECB
725   1480 26   F7              BNE   MOVST1
726   1482 30   1F      MVEXIT  LEAX  -1,X
727   1484 39                   RTS
728
729   1485 4D           CVTVR8  TSTA
730   1486 2A   0C              BPL   CVTVR1
731   1488 34   02              PSHS  A
732   148A 86   2D              LDA   #$2D
733   148C 8D   A9              BSR   MOVCHR
734   148E 35   02              PULS  A
735   1490 40                   NEGA
736   1491 50                   NEGB
737   1492 82   00              SBCA  #0
738   1494 6F   E2      CVTVR1  CLR   0,-S
739   1496 6F   E2              CLR   0,-S
740   1498 1F   41              TFR   S,X
741   149A 83   2710    CVTVR2  SUBD  #10000
742   149D 25   04              BCS   CVTVR3
743   149F 6C   84              INC   0,X
744   14A1 20   F7              BRA   CVTVR2
745   14A3 C3   2710    CVTVR3  ADDD  #10000
746   14A6 8D   BA              BSR   BIAS
747   14A8 83   03E8    CVTVR4  SUBD  #1000
748   14AB 25   04              BCS   CVTVR5
749   14AD 6C   84              INC   0,X
750   14AF 20   F7              BRA   CVTVR4
751   14B1 C3   03E8    CVTVR5  ADDD  #1000
752   14B4 8D   AC              BSR   BIAS
753   14B6 83   0064    CVTVR6  SUBD  #100
754   14B9 25   04              BCS   CVTVR7
755   14BB 6C   84              INC   0,X
756   14BD 20   F7              BRA   CVTVR6
757   14BF C3   0064    CVTVR7  ADDD  #100
758   14C2 8D   9E              BSR   BIAS
759   14C4 C0   0A      CVTVR8  SUBB  #10
760   14C6 25   04              BCS   CVTVR9
761   14C8 6C   84              INC   0,X
762   14CA 20   F8              BRA   CVTVR8
763   14CC CB   0A      CVTVR9  ADDB  #10
764   14CE 8D   92              BSR   BIAS
765   14D0 E7   84              STB   0,X
766   14D2 6C   01              INC   1,X
767   14D4 8D   0C              BSR   BIAS
768   14D6 32   62              LEAS  2,S
769   14D8 39                   RTS
770
771   14D9 1F   01      GETSTR  TFR   D,X
772   14DB EC   84              LDD   0,X
773   14DD 39                   RTS
774
775   14DE EF   62      NOT_EQ  STU   2,S
776   14E0 39                   RTS
777
778   14E1 CC   0000    LEN     LDD   #0000
779   14E4 6D   80      LEN1    TST   0,X+
780   14E6 27   03              BEQ   LEN2
781   14E8 5C                   INCB
782   14E9 20   F9              BRA   LEN1
783   14EB 39           LEN2    RTS
784
785   14EC A6   C0      LET     LDA   0,U+
786   14EE A7   80              STA   0,X+
787   14F0 27   03              BEQ   LET_EX
788   14F2 5A                   DECB
789   14F3 26   F7              BNE   LET
790   14F5 39           LET_EX  RTS
791
792   14F6 AE   62      MID     LDX   2,S
793   14F8 1F   13              TFR   X,U
794   14FA 4D                   TSTA
795   14FB 27   22              BEQ   RIGHT1
796   14FD 6D   86      MID_1   TST   0,X
797   14FF 27   1C              BEQ   RIGHT
798   1501 4A                   DECA
799   1502 27   04              BEQ   MID_2
800   1504 30   01              LEAX  1,X
801   1506 20   F5              BRA   MID_1
802   1508 5D           MID_2   TSTB
803   1509 27   12              BEQ   RIGHT
804   150B 8D   1E              BSR   MOVVR1
805   150D 20   14              BRA   RIGHT2
806
807   150F AE   62      LEFT    LDX   2,S
808   1511 5D           LEFT_1  TSTB
809   1512 27   0B              BEQ   RIGHT1
810   1514 A6   84              LDA   0,X
811   1516 27   07              BEQ   RIGHT1
812   1518 30   01              LEAX  1,X
813   151A 5A                   DECB
```

```
642  13E8 8D  02      PDATA   BSR   OUTEEE
643  13EA 20  F5              BRA   PDATA1
644
645  13EC 7E  CD10    OUTEEE  JMP   PUTCHR
646
647  13EF 86  3F      INPUT   LDA   #$3F
648  13F1 8D  F9              BSR   OUTEEE
649  13F3 86  20              LDA   #$20
650  13F5 8D  F5              BSR   OUTEEE
651  13F7 8D  30      INPUT1  BSR   INPUT8
652  13F9 C6  80              LDB   #$80
653  13FB 8D  CD15    INPUT2  JSR   GETCHR
654  13FE 81  CC00            CMPA  #$CRR
655  1401 26  09              BNE   INPDT3
656  1403 C1  80              CMPB  #$80
657  1405 27  0A              BEQ   INPDT4
658  1407 5C                  INCB
659  1408 30  1F              LEAX  -1,X
660  140A 20  EF              BRA   INPUT2
661  140C B1  CC01    INPUT3  CMPA  DL_CHR
662  140F 26  09              BNE   INPUT5
663  1411 8E  157F    INPUT4  LDX   #DELSTR
664  1414 8D  CB              BSR   PDATA1
665  1416 8D  0C              BSR   INPUT6
666  1418 20  D0              BRA   INPUT1
667  141A A7  84      INPUT5  STA   0,X
668  141C 81  0D              CMPA  #$0D
669  141E 26  09              BNE   INPUT7
670  1420 6F  80              CLR   0,X+
671  1422 9F  06              STX   BUFPNT
672  1424 8E  1586    INPUT6  LDX   #CRLF
673  1427 20  B8              BRA   PDATA1
674  1429 50          INPUT7  TSTB
675  142A 27  CF              BEQ   INPUT2
676  142C 30  01              LEAX  1,X
677  142E 5A                  DECB
678  142F 20  CA              BRA   INPUT2
679  1431 8E  06EE    INPUT8  LDX   #IO_BUF
680  1434 39                  RTS
681
682  1435 86  20      MOVSPC  LDA   #$20
683  1437 109E 06     MOVCHR  LDY   BUFPNT
684  143A A7  A0              STA   0,Y+
685  143C 6F  A4              CLR   0,Y
686  143E 109F 06             STY   BUFPNT
687  1441 39                  RTS
688
689  1442 8E  06EE    SETBUF  LDX   #IO_BUF
690  1445 9F  06              STX   BUFPNT
691  1447 39                  RTS
692
693  1448 80  E7      PSTRNG  BSR   INPUT8
694  144A A6  80      PSTNG1  LDA   0,X+
695  144C 27  04              BEQ   EXIT
696  144E 8D  9C              BSR   OUTEEE
697  1450 20  F8              BRA   PSTNG1
698  1452 39          EXIT    RTS
699
700  1453 80  F3      PST_CR  BSR   PSTRNG
701  1455 8E  1586    PCRLF   LDX   #CRLF
702  1458 A6  80      PCRLF1  LDA   0,X+
703  145A 81  04              CMPA  #4
704  145C 27  F4              BEQ   EXIT
705  145E 8D  8C              BSR   OUTEEE
706  1460 20  F6              BRA   PCRLF1
707
708  1462 34  02      BIAS    PSHS  A
709  1464 A6  84              LDA   0,X
710  1466 26  04              BNE   BIAS1
711  1468 6D  01              TST   1,X
712  146A 27  06              BEQ   BIAS2
713  146C 8A  30      BIAS1   ORA   #$30
714  146E 8D  C7              BSR   MOVCHR
715  1470 6C  01              INC   1,X
716  1472 6F  84      BIAS2   CLR   0,X
717  1474 35  02              PULS  A
718  1476 39                  RTS
719
720  1477 C6  80      MOVSTR  LDB   #$80
721  1479 A6  80      MOVST1  LDA   0,X+
722  147B 27  05              BEQ   MVEXIT
723  147D 80  B8              BSR   MOVCHR
724  147F 5A                  DECB
725  1480 26  F7              BNE   MOVST1
726  1482 30  1F      MVEXIT  LEAX  -1,X
727  1484 39                  RTS
```

```
728
729  1485 4D          CVTVAR  TSTA
730  1486 2A  0C              BPL   CVTVR1
731  1488 34  02              PSHS  A
732  148A 86  2D              LDA   #$2D
733  148C 8D  A9              BSR   MOVCHR
734  148E 35  02              PULS  A
735  1490 40                  NEGA
736  1491 50                  NEGB
737  1492 82  00              SBCA  #0
738  1494 6F  E2      CVTVR1  CLR   0,-S
739  1496 6F  E2              CLR   0,-S
740  1498 1F  41              TFR   S,X
741  149A 83  2710    CVTVR2  SUBD  #10000
742  149D 25  04              BCS   CVTVR3
743  149F 6C  84              INC   0,X
744  14A1 20  F7              BRA   CVTVR2
745  14A3 C3  2710    CVTVR3  ADDD  #10000
746  14A6 8D  BA              BSR   BIAS
747  14A8 83  03E8    CVTVR4  SUBD  #1000
748  14AB 25  04              BCS   CVTVR5
749  14AD 6C  84              INC   0,X
750  14AF 20  F7              BRA   CVTVR4
751  14B1 C3  03E8    CVTVR5  ADDD  #1000
752  14B4 8D  AC              BSR   BIAS
753  14B6 83  0064    CVTVR6  SUBD  #100
754  14B9 25  04              BCS   CVTVR7
755  14BB 6C  84              INC   0,X
756  14BD 20  F7              BRA   CVTVR6
757  14BF C3  0064    CVTVR7  ADDD  #100
758  14C2 8D  9E              BSR   BIAS
759  14C4 C0  0A      CVTVR8  SUBB  #10
760  14C6 25  04              BCS   CVTVR9
761  14C8 6C  84              INC   0,X
762  14CA 20  F8              BRA   CVTVR8
763  14CC CB  0A      CVTVR9  ADDB  #10
764  14CE 8D  92              BSR   BIAS
765  14D0 E7  84              STB   0,X
766  14D2 6C  01              INC   1,X
767  14D4 8D  8C              BSR   BIAS
768  14D6 32  62              LEAS  2,S
769  14D8 39                  RTS
770
771  14D9 1F  01      GETSTR  TFR   D,X
772  14DB EC  84              LDD   0,X
773  14DD 39                  RTS
774
775  14DE EF  62      NOT_BQ  STU   2,S
776  14E0 39                  RTS
777
778  14E1 CC  0000    LEN     LDD   #0000
779  14E4 6D  80      LEN1    TST   0,X+
780  14E6 27  03              BEQ   LEN2
781  14E8 5C                  INCB
782  14E9 20  F9              BRA   LEN1
783  14EB 39          LEN2    RTS
784
785  14EC A6  C0      LET     LDA   0,U+
786  14EE A7  80              STA   0,X+
787  14F0 27  03              BEQ   LET_EX
788  14F2 5A                  DECB
789  14F3 26  F7              BNE   LET
790  14F5 39          LET_EX  RTS
791
792  14F6 AE  62      MID     LDX   2,S
793  14F8 1F  13              TFR   X,U
794  14FA 4D                  TSTA
795  14FB 27  22              BEQ   RIGHT1
796  14FD 6D  84      MID_1   TST   0,X
797  14FF 27  1C              BEQ   RIGHT
798  1501 4A                  DECA
799  1502 27  04              BEQ   MID_2
800  1504 30  01              LEAX  1,X
801  1506 20  F5              BRA   MID_1
802  1508 5D          MID_2   TSTB
803  1509 27  12              BEQ   RIGHT
804  150B 8D  1E              BSR   MOVVR1
805  150D 20  14              BRA   RIGHT2
806
807  150F AE  62      LEFT    LDX   2,S
808  1511 5D          LEFT_1  TSTB
809  1512 27  0B              BEQ   RIGHT1
810  1514 A6  84              LDA   0,X
811  1516 27  07              BEQ   RIGHT1
812  1518 30  01              LEAX  1,X
813  151A 5A                  DECB
```

```
470
471  12B0 BD  13BC          JSR   NEXTX
472  12B3 20  CB            BLT   LN_450
473  12B5 26  03            BNE   LN_510
474  12B7 5D                TSTB
475  12B8 27  C6            BEQ   LN_450
476
477                 * 0510 GOTO 140
478
479  12BA 7E  10C3  LN_510  JMP   LN_140
480
481                 * 0520 LET D$=""
482
483  12BD 8E  000E  LN_520  LDX   #D_STR
484  12C0 BD  13DC          JSR   STNG20
485
486                 * 0530 FOR Y=1 TO LEN(C$(X))
487
488  12C3 B0  1391          JSR   Y_CDX
489
490                 * 0540 LET D$=D$+MID$(C$(X1,Y,1)
491
492  12C6 8E  000E  LN_540  LDX   #D_STR
493  12C9 CE  076F          LDU   #ST_BUF
494  12CC C6  50            LDB   #MAXLEN
495  12CE BD  152B          JSR   MOVVR1
496  12D1 32  7E            LEAS  -2,S
497  12D3 BD  140E          JSR   NOT_EQ
498  12D6 D6  09            LDB   X_VAR+1
499  12D8 86  50            LDA   #MAXLEN
500  12DA 3D                MUL
501  12DB C3  037E          ADDD  #C_STR-MAXLEN
502  12DE BD  14D9          JSR   GETSTR
503  12E1 C6  50            LDB   #MAXLEN
504  12E3 BD  152B          JSR   MOVVR1
505  12E6 D6  0B            LDB   Y_VAR+1
506  12E8 34  04            PSHS  B
507  12EA C6  01            LDB   #1
508  12EC 35  02            PULS  A
509  12EE BD  14F6          JSR   MID
510  12F1 CE  076F          LDU   #ST_BUF
511  12F4 8E  000E          LDX   #D_STR
512  12F7 BD  1307          JSR   STNG_2
513
514                 * 0550 LET D$=D$+" "
515
516  12FA BD  74            BSR   DD_SPC
517
518                 * 0560 IF MID$(D$,LEN(D$)-1,1)<>" " THEN 570
519
520  12FC CE  076F          LDU   #ST_BUF
521  12FF 32  7E            LEAS  -2,S
522  1301 BD  140E          JSR   NOT_EQ
523  1304 8E  000E          LDX   #D_STR
524  1307 C6  50            LDB   #MAXLEN
525  1309 BD  152B          JSR   MOVVR1
526  130C 32  7E            LEAS  -2,S
527  130E BD  140E          JSR   NOT_EQ
528  1311 8E  000E          LDX   #D_STR
529  1314 C6  50            LDB   #MAXLEN
530  1316 BD  152B          JSR   MOVVR1
531  1319 AE  E4            LDX   0,S
532  131B BD  14E1          JSR   LEN
533  131E BD  151D          JSR   RIGHT
534  1321 83  0001          SUBD  #$0001
535  1324 34  04            PSHS  B
536  1326 C6  01            LDB   #1
537  1328 35  02            PULS  A
538  132A BD  14F6          JSR   MID
539  132D 8E  076F          LDX   #ST_BUF
540  1330 BD  1546          JSR   EQUALS
541  1333 0F  04            STU   X_TEMP
542  1335 8E  156B          LDX   #SPACE
543  1338 BD  1529          JSR   MOVVAR
544  133B 8E  076F          LDX   #ST_BUF
545  133E BD  154A          JSR   IF
546  1341 24  02            BCC   LN_570
547
548                 * 0565 LET D$=D$+" "
549
550  1343 BD  28            BSR   DD_SPC
551
552                 * 0570 NEXT Y
553
554  1345 BD  77    LN_570  BSR   NEXTY
555  1348 1020 FF7B         LBLT  LN_540
556  134C 26  05            BNE   LN_580
557  134E 5D                TSTB
558  134F 1027 FF74         LBEQ  LN_540
559
560                 * 0580 LET C$(X1)=D$
561
562  1353 D6  09    LN_580  LDB   X_VAR+1
563  1355 86  50            LDA   #MAXLEN
564  1357 3D                MUL
565  1358 C3  037E          ADDD  #C_STR-MAXLEN
566  135B 8E  000E          LDX   #D_STR
567  135E BD  6C            BSR   STNG_1
568
569                 * 0590 RETURN
570
571  1360 39                RTS
572
573                 * 0600 STOP
574
575  1361 7E  CD03  LN_600  JMP   WARMS
576
577                 * Start of run-time package
578
579  1364 D6  09    LBDX    LDB   X_VAR+1
580  1366 86  50            LDA   #MAXLEN
581  1368 3D                MUL
582  1369 C3  005E          ADDD  #B_STR-MAXLEN
583  136C 8E  005E          LDX   #E_STR
584  136F 20  5B            BRA   STNG_1
585
586  1371 8E  000E  DD_SPC  LDX   #D_STR
587  1374 CE  076F          LDU   #ST_BUF
588  1377 C6  50            LDB   #MAXLEN
589  1379 BD  152B          JSR   MOVVR1
590  137C 8E  1560          LDX   #SPACE
591  137F BD  1529          JSR   MOVVAR
592  1382 CE  076F          LDU   #ST_BUF
593  1385 8E  000E          LDX   #D_STR
594  1388 20  46            BRA   STNG_2
595
596  138A CC  0001  Y_CDX   LDD   #1
597  138D DD  0A            STD   Y_VAR
598  138F D6  09            LDB   X_VAR+1
599  1391 86  50            LDA   #MAXLEN
600  1393 3D                MUL
601  1394 C3  037E          ADDD  #C_STR-MAXLEN
602  1397 BD  1409          JSR   GETSTR
603  139A CE  076F          LDU   #ST_BUF
604  139D C6  50            LDB   #MAXLEN
605  139F BD  1528          JSR   MOVVR1
606  13A2 8E  076F          LDX   #ST_BUF
607  13A5 B0  14E1          JSR   LEN
608  13A8 D0  02            STD   NEXT_Y
609  13AA 39                RTS
610
611  13AB CC  0001  X_12_Z  LDD   #1
612  13AE DD  08            STD   X_VAR
613  13B0 DC  0C            LDD   Z_VAR
614  13B2 DD  00            STD   NEXT_X
615  13B4 39                RTS
616
617  13B5 DC  08    NEXTX   LDD   X_VAR
618  13B7 C3  0001          ADDD  #$0001
619  13BA DD  08            STD   X_VAR
620  13BC 93  00            SUBD  NEXT_X
621  13BE 39                RTS
622
623  13BF DC  0A    NEXTY   LDD   Y_VAR
624  13C1 C3  0001          ADDD  #$0001
625  13C4 DD  0A            STD   Y_VAR
626  13C6 93  02            SUBD  NEXT_Y
627  13C8 39                RTS
628
629  13C9 8E  156C  STNG_0  LDX   #ZERO
630  13CC 1F  13    STNG_1  TFR   X,U
631  13CE 1F  01            TFR   D,X
632  13D0 C6  50    STNG_2  LDB   #MAXLEN
633  13D2 7E  14EC          JMP   LET
634
635  13D5 CE  156C  STNG20  LDU   #ZERO
636  13D8 20  F6            BRA   STNG_2
637
638  13DA A6  80    PDATA1  LDA   0,X+
639  13DC 81  04            CMPA  #4
640  13DE 26  01            BNE   PDATA
641  13E0 39                RTS
```

```
        blo rdfull don't have it
        tfr d,y length to Y
        ldx endpt,U buffer address
        lda 0,s path
        OS9 I$READLN read a line
        bcs rdret EOF or error
*
        tfr y,d actual length to D
        leax d,x point past line just read
        bra rd10 and read some more
*
rdfull  clrb clear carry
rdret   puls a,pc return
*
*       Write buffer to output file
*       Input:  A = path number, X = buffer address
*       Output: carry clear if all OK
*               carry set, B = error code if error
*
writeit pshs a save path number
wr10    stx outpt,U save buffer pointer
        ldd endpt,U
        subd outpt,U done yet?
        bls wrdone yes
*
        tfr d,y (max) length to Y
        lda 0,s path number
        OS9 I$WRITLN write a line
        bcs wrret exit if error
*
        tfr y,d actual length to D
        leax d,x point past line just written
        bra wr10
*
wrdone  clra all OK
wrret   puls a,pc return
        emod
modsz   equ * end of program
        end
```

EOF

Gespac

GESPAC INTRODUCES A 68020 CPU BOARD FOR THE G-64 BUS

WESCON 86, BOOTH 3167, ANAHEIM, CA, NOVEMBER 18, 1986--GESPAC announces a new processor board based around Motorola's 68020, 32-bit microprocessor. The GESMPU-20 is built on a standard single height Eurocard of 4 by 6.25 inches, and is fully compatible with the standard G-64 bus.

On just 25 square inches, the GESMPU-20 includes a 68020 processor running at 12.5 MHz (16 MHz optional), and a 68881 floating point coprocessor. The 68020 is one of the most advanced and most popular microprocessor in the market, with such features as virtual machine support, on-chip instruction cache, and operating speed up to 5 MIPS. The arithmetic coprocessor is optional and supports the full IEEE floating point standard proposal P754.

The GESMPU-20 comes equipped with four 32 pin sockets capable of supporting EPROMs with densities from 64 Kbit to 1 Mbit. Using the highest density devices, the GESMPU-20 can hold up to 512 Kilobytes of EPROM. The board also comes equipped with 256 Kilobytes of fast access, CMOS RAM. These RAM devices have an access time fast enough to permit operation with zero wait states. The GESMPU-20 is available with an optional 512 Kilobytes of RAM, bringing the total memory capacity of the board to 1 Megabytes of EPROM/RAM.

All memory transfers on the board are executed in a fast 32-bit mode. The GESMPU-20 can access up to 32 Megabytes of external memory and standard I/O through its G-64 bus interface. The G-64 bus is a processor independent, non-multiplexed 16-bit bus, specially geared towards midrange industrial control applications. Since the G-64 bus is a 16-bit bus, the dynamic bus sizing feature of the 68020 microprocessor is used to exchange data through the G-64 bus.

Because of the very compact size and high reliability of the single Eurocard form factor and DIN connector, the GESMPU-20 is ideally suited for a wide range of applications such as industrial process control, portable test equipment, instrumentation, and navigation computers.

The GESMPU-20 is well supported by software tools and application packages from GESPAC. The GESMPU-20 runs the OS-9, real time, multi-tasking disk operating system, for which a Pascal, C and Basic compiler, and an editor/assembler are available. GESPAC also has a family of device drivers for its family of graphics controllers, and local area network.

This software offering, combined with over 150 G-64 bus system components available from GESPAC, makes the GESMPU-20 the control element of a "total solution" package for system integrators.

The GESMPU-20 is available in sample quantities today. The board equipped with 256 Kilobyte of RAM is priced at $995 in low volume orders. Production of the board is scheduled for January 1987.

For more information contact: Cosma Pabouctsidis
GESPAC, Inc.
50 W. Hoover Ave.
Mesa, AZ 85202
(602) 962-5559

Classifieds

As submitted - No Guarantees

DAISY WHEEL PRINTERS
Qume Sprint 9 - $900 Qume Sprint 5 - $800 TEC FP1500 - $800

Winchester 10 Megabyte Drive - Seagate Model #412 $275.

3 - Dual 8" drive enclosure with power supply. New in box. $125 each.

5 - Siemens 8" Disk Drives. $100 each.

Tano Outpost II, 56K, 2 5" DSDD Drives, FLEX, MUMPS $495.

TELETYPE Model 43 PRINTER - with serial (RS232) interface and full ASCII keyboard. $250 ready to run.

SWTPC S/09 with Motorola 128K RAM, 1-MPS2, 1-Parallel Port, MP-09 CU Card - $900 complete.

CDS-1- 20 Meg Hard Disk System with controller $500.

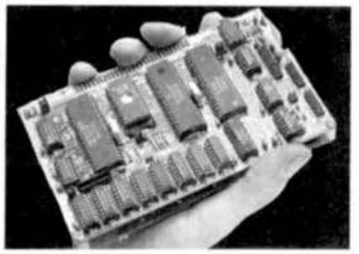
(615) 842-4600 M-F 9 AM to 5 PM EST

•••

SWTPc-S+, W/256K, 46 MG. Winchester Drive, One X-12 Terminal, Four QUME Terminals, UDRI Data Base, Twelve Serial And Four Parallel Ports. Will transfer rights of software incl Stylograph and G/L. Invested $30K sacrifice - $7500/Offer. (206) 898-2481, Lynne.

•••

GIMIX DMA III Disk controllers, 7 Boards Available, Used/Tested, $175.00 ea/swap.
(916)393-9912.



512K RAM Expansion

Compact Flexible 6809 Computer

The new ST-2900 system — a complete 64K small business or hobbyist computer is only one of its many possible configurations. Among its features are:

- Small enough to hold in your hand! (Eurocard size 3.9" x 6.3")
- Three board "system" for greater versatility than single board computers.
- CPU Board — powerful 6809E processor, 16K or 64K RAM, 1K-32K EPROM, 2 RS232 serial ports with software programmable baud rates, 16 bit counter/timer. Run the CPU board all by itself, or plug your own custom board or our FDC board and/or RAM-512 board into the expansion connector.
- FDC Board — double-sided/double-density floppy disk controller with adjustment free digital data separator and write precompensation, 2 8-bit parallel ports, 2 16-bit counter/timers, prototyping area.
- **RAM-512 Board** — 524 288 bytes of RAM on a 4.15" x 6.3" board! Low power. Includes RAM-Disk software for FLEX/STAR-DOS or OS-9.
- **FLEX, STAR-DOS,** and **OS-9** supported — **software selectable.**
- OS-9 Conversion Package lets you use the low cost Radio Shack CoCo version of OS-9 on our ST-2900 system. Save $131 off the suggested list price of OS-9! **No programming is involved.** Supports CoCo OS-9, standard OS-9, and MIZAR OS-9/68K disk formats. Compatible with PC-XFER to let you **read/write/format MS-DOS disket**
- CPU bare board plus EPROM $45 — OS-9 Conversion Package $49
  FDC bare board $38 — FLEX Conversion Package $29
  RAM-512 board A&T **(w/o RAM) $299** — CPU + FDC + OS-9 Conversion $119
  CPU + FDC board set assembled and tested $329
- Add $5 shipping/handling ($10 overseas). These prices are in U.S. funds. Canadian orders: call or write for prices. Terms: check, money order, VISA.

Trademarks: FLEX — Technical Systems Consultants; OS-9 — Microware & Motorola; MS-DOS — Microsoft

Source code for ST-2900 OS-9 device drivers now available $99

SARDIS TECHNOLOGIES

Call or write for free catalog and complete price list
(604) 255-4485

2261 E. 11th Ave. Vancouver, B.C., Canada V5N 1Z7

Hard Disk Subsystem for SS-50 Computers

THIS PROVEN SUBSYSTEM ADDS HARD DISK SPEED AND STORAGE CAPACITY TO YOUR COMPUTER YET REQUIRES ONLY ONE SS-30 SLOT. SOFTWARE (WITH SOURCE) IS INCLUDED FOR YOUR CHOICE OF FLEX9* OR SK*DOS*, OS-9* LEVEL I OR LEVEL II, OR OS-9 68K OPERATING SYSTEMS. THE SOFTWARE HONORS ALL OPERATING SYSTEM CONVENTIONS. THE SOFTWARE IS DESIGNED FOR THE XEBEX S1410 CONTROLLER INTERFACING TO ANY HARD DISK DRIVE THAT CONFORMS TO THE ST506 STANDARD. FOUR SUBSYSTEMS ARE AVAILABLE:

1) 27 MB (FORMATTED) CONTROL DATA CORPORATION WREN HARD DISK, XEBEC S1410A CONTROLLER, SS-30 INTERFACE CARD, ALL CABLES, AND SOFTWARE FOR $2850;
2) 7.3 MB (FORMATTED) TANDON TM-603 HARD DISK, REST SAME AS ABOVE FOR $895;
3) NO HARD DISK, REST SAME AS ABOVE FOR $600; AND
4) SS-50 INTERFACE CARD AND SOFTWARE FOR $200.

ALL PRICES INCLUDE SHIPPING. WE ACCEPT VISA AND MASTERCARD WITHOUT ADDING A SURCHARGE. TEXAS RESIDENTS MUST ADD SALES TAX. THE SUBSYSTEM MAY BE MOUNTED WITHIN YOUR COMPUTER CHASSIS OR IN A SEPARATE ENCLOSURE WITH POWER SUPPLY. PLEASE WRITE OR PHONE (INCLUDE YOUR DAY AND EVENING PHONE NUMBERS) FOR MORE INFORMATION. WE WILL RETURN NORTH AMERICA CALLS SO THAT ANY DETAILED ANSWERS WILL BE AT OUR EXPENSE.

WELLWRITTEN™ ENTERPRISES

P.O. BOX 9802 - 845
AUSTIN, TEXAS 78766

VISA — *** (512) 241-6530 *** — MasterCard

FLEX IS A TRADEMARK OF TECHNICAL SYSTEMS CONSULTANTS, INC.
SK*DOS IS A TRADEMARK OF STAR-KITS
OS-9 IS A TRADEMARK OF MICROWARE AND MOTOROLA

SK★DOS™

The Generic DOS™ for 68000 applications in

- ★ Industrial Control
- ★ Business Use
- ★ Educational Computing
- ★ Scientific Computing
- ★ Number Crunching
- ★ Dedicated Systems
- ★ Turnkey Systems
- ★ Data Collection
- ★ Single board Computers
- ★ Bus oriented Computers
- ★ Graphics Workstations
- ★ One of a kind Systems
- ★ Advanced Hobbyist Use

SK★DOS is a single user disk operating system for computers using Motorola 32 bit CPUs such as the 68008, 68000, 68010, and 68020. It provides the power of a full DOS, yet is simple and easy to use, and will run on systems from 32K to 16 megabytes. Because SK★DOS is easily implemented on a new system, we call it "The Generic DOS" which allows programs written for one system to be run on many others.

SK★DOS comes with over 40 commands and system programs, including a 6809 emulator which allows 68K SK★DOS to run application programs and languages developed for 6809 SK★DOS and other systems. Assemblers, editors, and higher level language support are available from third party software vendors and through public domain software.

SK★DOS is available for single copy or dealer sales, as well as OEM licensing. Single copies cost $125 (inquire as to available systems). Extremely attractive OEM licensing terms are also available. An optional Configuration Kit contains a detailed Configuration Manual and two disks of source code for system adaptation, including source code for a system monitor/debug ROM and other programs useful for adapting SK★DOS to new systems.

SK★DOS *is available from*

STAR-K
SOFTWARE SYSTEMS CORPORATION
P.O. BOX 209 · MT. KISCO, NY 10549 · 914/241-0287
TELEX 510601677 4

Call for Software:

68000, C, Basic09 Sculptor

We are receiving calls and letters from numerous sources, including users, business and others looking for OS-9 68000 software; applications, etc.

Many of you have developed software that with little change could be adopted for others. If you are interesting in selling it, please let us know. There is a growing market out there now. Get in on the ground floor!

If you can use additional income and have something that might be of interest, call and talk to Larry or Don.

S.E. MEDIA Division - CPI
POB 849
Hixson, TN 37343

Telephone (615) 842-6809
Telex (510) 600-6630

OS-9 UniFLEX
MUSTANG-020, 68020, 68881 AND MORE
HANDS-ON EXPERIENCE

The DATA-Comp Division of Computer Publishing Corporation announces their new and innovative HANDS-ON 68020 computer familiarization two day event. A chance to TRY BEFORE YOU BUY!

For two full days (Monday through Friday - excluding legal holidays) each participant will be furnished the exclusive use of a 68020 computer (MUSTANG-020). Each system will have available native C compilers, BASIC, assembler and other high level languages. Each system will be equipped with the Motorola MC 68881 math co-processor, where applicable.

Each demonstration room will contain not more than two work stations. Each system will be equipped with floppy disk, 20 megabyte winchester technology hard disk, and 2 megabyte of RAM. RAM is partitioned as 690K bytes of RAM disk and 1.2 megabyte of user RAM space.

Participants are encouraged to bring along any source level projects, for evaluation, in C, BASIC or assembler. Call for availibility of other HHLs.

Although this is not a training seminar, Data-Comp personnel are available for assistance and consultation. This event is scheduled for hands-on evaluations of the 68020 CPU, 68881 math co-processor and MUSTANG-020 system, operating in a functional environment.

Transportation to and from the airport and hotel/motel will be provided. Lunch provided both days. Chattanooga airport is serviced by American, Delta, Republic and other airlines.

COST

One person - $375.00

Two persons - $595.00

* Motel single $22.00, double $26.00
Includes satellite TV - convenient to food and shopping

DATA-COMP

A Division of
Computer Publishing, Inc.

5900 Cassandra Smith Road
Hixson, Tn 37343
Telephone 615 842-4600
Telex 510 600-6630

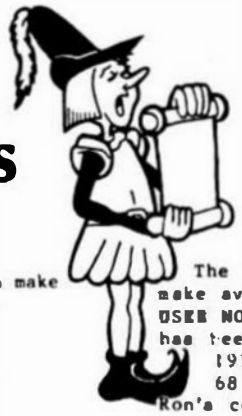
Systems available for both OS-9 and UniFLEX. Reservation should be made 15 days in advance. Attendee should initially indicate OS-9, UniFLEX or both. Special facilities available on request. Please write or call for additional information.

NOTE: Both OS-9 and UniFLEX are Unix type operating systems. Each as been enhanced in some aspect or another. Prospective attendees should have some working knowledge or experience with one of these operating systems, to gain full benefit of the session. However, a newcomer will find that it is a simple matter to be fairly proficient in using these systems in the allocated time. Special system instruction available on request. Call or write.

* Hotel/Motel cost are separate cost, not included in the basic cost shown.

THE 6800-6809 BOOKS

..HEAR YE.....HEAR



OS-9™ User Notes

By: Peter Dibble

The publishers of 68' Micro Journal are proud to make available the publication of Peter Dibbles **OS9 USER NOTES**

Information for the BEGINNER to the PRO, Regular or CoCo OS9

Using OS9
HELP, HINTS, PROBLEMS, REVIEWS, SUGGESTIONS, COMPLAINTS, OS9 STANDARDS, Generating a New Bootstrap, Building a new System Disk, OS9 Users Group, etc.

Program interfacing to OS9
DEVICE DESCRIPTORS, DIRECTORIES, "FORKS", PROTECTION, "SUSPEND STATE", "PIPES", "INPUT/OUTPUT SYSTEM", etc.

Programming Languages
Assembly Language Programs and Interfacing; Basic09, C, Pascal, and Cobol reviews, programs, and uses; etc.

Disks Include
No typing all the Source Listings in. Source Code and, where applicable, assembled or compiled Operating Programs. The Source and the Discussions in the Columns can be used "as is", or as a "Starting Point" for developing your OWN more powerful Programs. Programs sometimes use multiple Languages such as a short Assembly Language Routine for reading a Directory, which is then "piped" to a Basic09 Routine for output formatting, etc.

BOOK $9.95

Typeset -- w/ Source Listings
(3-Hole Punched; 8 x 11)
Deluxe Binder - - - - - - - - - - $5.50

All Source Listings on Disk

1-8" SS, SD Disk - - - - $14.95
2-5" SS, DD Disks - - - $24.95

FLEX™ USER NOTES

By: Ronald Anderson

The publishers of 68 MICRO JOURNAL are proud to make available the publication of Ron Anderson's **FLEX USER NOTES**, in book form. This popular monthly column has been a regular feature in 68' MICRO JOURNAL SINCE 1979. It has earned the respect of thousands of 68 MICRO JOURNAL readers over the years. In fact, Ron's column has been described as the 'Bible' for 68XX users, by some of the world's leading microprocessor professionals. The most needed and popular 68XX book available. Over the years Ron's column has been one of the most popular in 68 MICRO JOURNAL. And of course 68 MICRO JOURNAL is the most popular 68XX magazine published.

Listed below are a few of the **TEXT** files included in the book and on diskette.

All TEXT files in the book are on the disks.

LOGO.C1	File load program to offset memory — ASM PIC
MEMOVEC1	Memory move program — ASM PIC
DUMP.C1	Printer dump program — uses LOGO — ASM PIC
SUBTEST.C1	Simulaton of 6800 code to 6809, show differences — ASM
TERMEM.C2	Modem input to disk (or other port input to disk) — ASM
M.C2	Output a file to modem (or another port) — ASM
PRINT.C3	Parallel (enhanced) printer driver — ASM
MODEM.C2	TTL output to CRT and modem (or other port) — ASM
SCIPKG.C1	Scientific math routines — PASCAL
U.C4	Mini-monitor, disk resident, many useful functions — ASM
PRINT.C4	Parallel printer driver, without PFLAG — ASM
SET.C5	Set printer modes — ASM
SETBAS1.C5	Set printer modes — A-BASIC

NOTE: .C1,.C2, etc.=Chapter 1, Chapter 2, etc.

**Over 30 TEXT files included is ASM (assembler)-PASCAL-PIC (position independent code) TSC BASIC-C, etc.

Book only: **$7.95** + $2.50 S/H

With disk: 5" **$20.90** + $2.50 S/H

With disk: 8" **$22.90** + $2.50 S/H

Shipping & Handling $3.50 per Book, $2.50 per Disk set
Foreign Orders Add $4.50 Surface Mail
or $7.00 Air Mail
If paying by check - Please allow 4-6 weeks delivery
• All Currency in U.S. Dollars

Continually Updated In 68 Micro Journal Monthly

VISA

Computer Publishing Inc.
5900 Cassandra Smith Rd.
Hixson, TN 37343

(615) 842-4601
Telex 5106006630

"FLEX is a trademark of Technical Systems Consultants
"OS9 is a trademark of Microware and Motorola
"68' Micro Journal is a trademark of Computer Publishing Inc.

AVAILABLE NOW!

PLμS-68k (PL/9 for the 68000) running under FLEX™

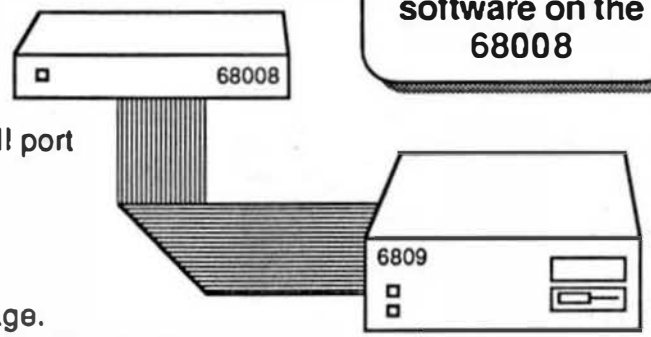
- Built-in screen editor
- Built-in source-level debugger
- Byte, Integer and Long variables
- Single-pass compiler
- Direct source to object
- Compiles over 1000 lines/min
- Runs on any FLEX™ system with a spare PIA port

Develop 68000 software on your FLEX™ system. No second computer required!

The second processor module (included):

- 10MHz 68008 CPU
- 128k bytes RAM
- Case and power supply
- Plugs into Windrush UPROM-III port

Run FLEX™ software on the 68008



Other software included:

- Program loader
- 68000 FLEX™ interface package.
- Comprehensive System Monitor with source
- Hex-Binary-Hex Conversion Routines

Development is currently under way of a version for OS/9-68000. The package price includes a free copy of the OS/9 version when it becomes available.

$999
Complete

For further information, phone or write:

Worstead Laboratories
North Walsham
Norfolk NR28 9SA
England

Tel (44) 692 404086
Telex 975548 WMICRO G

WINDRUSH
Micro Systems Ltd.

GMX Micro-20 prices

MICRO 20 (12.5 MHz).....$2565.00
MICRO 20 (16.67 MHz).....$2895.00
8 PORT RS232 BOARD SET (S8C-8S).....$ 498.00
PROTOTYPING BOARD (S8C-MW).....$ 75.00
BACK PANEL PLATE (8PP-PC).....$ 44.00
I/O BUS ADAPTER (S8C-8A).....$ 195.00

QUANTITY DISCOUNTS ARE AVAILABLE ON THE ABOVE ITEMS AS FOLLOWS:4-9,LESS 5%; 10-24,LESS 10%; 25-99,LESS 20%; 100 UP,LESS 30%.

MC68881RC12.....$ 295.00
MC68881RC16.....$ 395.00
S8C ACCESSORY PACKAGE (M20-AP).....$1690.00
for other configurations and options, contact GMX.
MOTOROLA 68020 USERS MANUAL.....$ 18.00
MOTOROLA 68881 USERS MANUAL.....$ 18.00

TO ORDER BY MAIL: SEND CHECK OR MONEY ORDER OR USE YOUR VISA OR MASTER CHARGE. Please allow 3 weeks for personal checks to clear. U.S. orders add $5 handling if order is under $200.00. Foreign orders add $10 handling if order is under $200.00. Foreign orders over $200.00 will be shipped via Emery Air Freight COLLECT, and we will charge no handling. All orders must be prepaid in U.S. funds. Please note that foreign checks have been taking about 8 weeks for collection so we would advise wiring money, or checks drawn on a bank account in the U.S. Our bank is the Continental Illinois National Bank of Chicago, 231 S. LaSalle Street, Chicago, IL 60693, account number 73-32033.

GMX
1337 WEST 37th PLACE
CHICAGO, ILLINOIS 60609
(312) 927-5510 • TWX 910-221-4055

GMX S-50 BUS prices

68020 SYSTEM

For the user who appreciates the need for a bus structured system using STATIC RAM and powered by a ferro resonant constant voltage transformer, DMA transfers, high speed MMU, we have the UNIFLEX-VM 68020 development system.
The system CPU provides protection to the system and other users from crashes caused by defective user programs.
The system's intelligent serial I/O processor boards significantly reduce system overhead by handling routine I/O functions.
The UniFLEX VM Operating System is a demand-paged, virtual memory operating system written in 68020 Assembler code for compactness and efficiency. It allows up to 4 Megabytes of Virtual Memory per user. All systems include 1MB of static RAM, one 3-port Intelligent Serial I/O board, DMA Controllers, a 5" 80 track floppy drive.

PRICES

#020 UniFLEX VM with 25MB HD.....$10,980.20
#020 UniFLEX VM with 85MB HD.....$12,480.20

YOU CAN EXPAND THESE 020 SYSTEMS WITH:
60MB STREAMER.....$ 2,400.00
REMOVABLE PACK DRIVE.....$ 1,200.00

INTELLIGENT I/OS
#14 3 Port Serial-30 Pin.....$ 498.14
#13 4 Port Serial-50 Pin.....$ 618.13
#12 Parallel-50 Pin.....$ 538.12

CABLE SETS FOR I/OS
95 Cable Sets Specify Card.....$ 24.95
51 Cent. D.P. Cable for #12 & #44....$ 34.51
53 Cent. Cable Set.....$ 36.53

6809 SYSTEM

The number 39 systems include:#05 CPUwDAT; #19 Classy Chassis; 256K Static RAM; a # 43 2 port serial card & cables; #68 DMA Controller; all necessary cables, power regulators, and filler plates;

System # 39 OS-9 GMX II Dual 80 DSDD...$ 2,998.39
" w19MB.....$ 4,698.39
w72MB.....$ 6,298.39

The Software included in this System: GMXBUG monitor; FLEX; and OS-9 GMXII. You can software select either FLEX or OS-9. Also includes OS-9 Editor, Assembler, Debugger, BASIC-09, RUNB, RMS, DD, and GMX-VDISK for FLEX,

System # 39 UniFLEX w25MB.....$ 4,698.39
" w85MB.....$ 6,298.39

The UniFLEX Operating System is included.

6809 SYSTEMS USING THE GIMIX III CPU & INTELLIGENT I/O PROCESSOR BOARDS

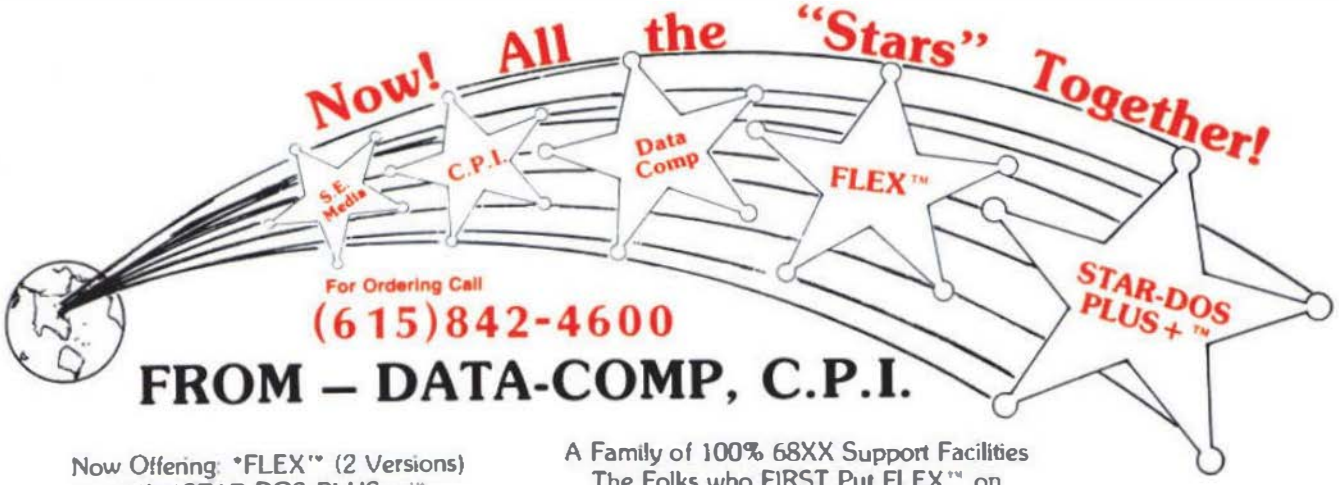
These System include: GMX6809 CPU III; one #11 3 port intelligent serial I/O & Cables; #19 Classy Chassis; 256K Static RAM; #68 DMA controller; all necessary cables, power regulators, and filler plates.

System # 79 OS9 GMX III Dual 80 DSDD...$ 4,498.79
" w25MB.....$ 6,498.79
" w85MB.....$ 7,998.79

The # 79 System Software includes: OS9 GMXIII; OS9 Editor, Assembler, Debugger, BASIC 09, RUNB, RMS, DD, RAMdisk, O-FLEX; GMXBUG; FLEX. The GMX Support ROM and the hardware CRC board are exclusive features included in this system.

System # 89 UniFLEX III w25MB.....$ 6,798.39
w85MB.....$ 8,298.39

The UniFLEX GMX III Operating System is included.



FROM – DATA-COMP, C.P.I.

Now Offering: *FLEX™ (2 Versions)
AND *STAR-DOS PLUS+™

A Family of 100% 68XX Support Facilities
The Folks who FIRST Put FLEX™ on
The CoCo

FLEX-CoCo Sr.
with TSC Editor
TSC Assembler
Complete with Manuals
Reg. $250.00 **Only $79.95**

STAR-DOS PLUS+
- Functions Same as FLEX
- Reads - writes FLEX Disks
- Run FLEX Programs
- Just type: Run "STAR-DOS"
- Over 300 utilities & programs to choose from.

$34.95

FLEX-CoCo Jr.
without TSC Editor & Assembler
$49.95

PLUS

ALL VERSIONS OF FLEX & STAR-DOS• INCLUDE

+ Read-Write-Dir RS Disk
+ Run RS Basic from Both
+ More Free Utilities
+ External Terminal Program
+ Test Disk Program
+ Disk Examine & Repair Program
+ Memory Examine Program
+ Many Many More!!!

TSC Editor
Reg $50.00
NOW $35.00

TSC Assembler
Reg $50.00
NOW $35.00

CoCo Disk Drive Systems

2 THINLINE DOUBLE SIDED DOUBLE DENSITY DISK DRIVES SYSTEM WITH POWER SUPPLY,CABINET,DISK DRIVE CABLE,J&M NEW DISK CONTROLLER JFD-CP WITH J-DOS,RS-DOS OPERATING SYSTEMS.	$469.95
* Specify What CONTROLLER You Want J&M, or RADIO SHACK	
THINLINE DOUBLE SIDED DOUBLE DENSITY 40 TRACKS	$129.95

Verbatim Diskettes

Single Sided Double Density	$ 24.00
Double Sided Double Density	$ 24.00

Controllers

J&M JFD-CP WITH J-DOS	$139.95
WITH J-DOS, RS-DOS	$159.95
RADIO SHACK 1.1	$134.95
RADIO SHACK Disk CONTROLLER 1.1	$134.95

Disk Drive Cables

Cable for One Drive	$ 19.95
Cable for Two Drives	$ 24.95

MISC

64K UPGRADE FOR C,D,E,P, AND COCO 11	$ 29.95
RADIO SHACK BASIC 1.2	$ 24.95
RADIO SHACK DISK BASIC 1.1	$ 24.95
DISK DRIVE CABINET FOR A SINGLE DRIVE	$ 49.95
DISK DRIVE CABINET FOR TWO THINLINE DRIVES	$ 69.95

PRINTERS

EPSON LX-80	$289.95
EPSON MX-70	$325.95
EPSON MX-100	$495.95

ACCESSORIES FOR EPSON

8148 2K SERIAL BOARD	$ 89.95
8149 32K EXPAND TO 128K	$169.95
EPSON MX-RX-80 RIBBONS	$ 7.95
EPSON LX-80 RIBBONS	$ 5.95
TRACTOR UNITS FOR LX-80	$ 39.95
CABLES & OTHER INTERFACES CALL FOR PRICING	

DATA-COMP
5900 Cassandra Smith Rd.
Hixson, TN 37343

MasterCard VISA

SHIPPING
USA ADD 2%
FOREIGN ADD 5%
MIN. $2.50

(615)842-4600
For Ordering
Telex 5106006630

Introducing

S - 50 BUS / 68XX

Board and/or Computer
Terminals-CRTs-Printers
Disk Drives-etc.

REPAIRS

This

Not This

NOW AVAILABLE TO ALL S50/68XX USERS

The Data-Comp Division of CPI is proud to announce the availability of their service department facilities to **'ALL'** S50 Bus and 68XX users. Including all brands, SWTPC - GIMIX - SSB - HELIX and others, including the single board computers. *Please note that kit-built components are a special case, and will be handled on an individual basis, if accepted.

1. If you require service, the first thing you need to do is call the number below and describe your problem and *confirm a Data-Comp service & shipping number*! This is very important, Data-Comp will not accept or repair items not displaying this number! Also we cannot advise or help you troubleshoot on the telephone, we can give you a shipping number, but **NO** advice! Sorry!

2. All service shipments must include both a minimum $40.00 estimate/repair charge and pre-paid return shipping charges (should be same amount you pay to ship to Data-Comp).

3. If you desire a telephone estimate after your repair item is received, include an additional $5.00 to cover long distance charges. Otherwise an estimate will be mailed to you, if you requested an estimate. Estimates *must be requested*. Mailed estimates slow down the process considerably. However, if repairs are not desired, after the estimate is given, the $40.00 shall constitute the estimate charge, and the item(s) will be returned unrepaired providing sufficient return shipping charges were included with the item to be serviced. Please note that estimates are given in dollar amounts only.

4. Data-Comp service is the oldest and most experienced general S50/68XX service department in the world. We have over $100,000.00 in parts in stock. We have the most complete set of service documents for the various S50/68XX systems of anyone - **YET, WE DO NOT HAVE EVERYTHING!** But we sure have more than anyone else. We repair about 90% of all items we receive. Call for additional information or shipping instructions.

DATA-COMP
5900 Cassandra Smith Rd.
Hixson, TN 37343

(615)842-4607
Telex 5106006630